Contractor Report 2005-03

Intelligent Terrain Analysis and Tactical Support System (ITATSS) for Unmanned Ground Vehicles

Randolph M. Jones
Soar Technology, Inc.

Ron Arkin
Georgia Institute of Technology

Nahid Sidki
Science Applications International Corporation

April 2005

United States Army Research Institute for the Behavioral and Social Sciences

REPORT DOCUMENTATION PAGE

1. REPORT DATE (dd-mm-yy) April 2005	2. REPORT TYPE Final	3. DATES COVERED (from. . . to) August 2004 to February 2005

4. TITLE AND SUBTITLE Intelligent Terrain Analysis and Tactical Support System (ITATSS) for Unmanned Ground Vehicles	5a. CONTRACT OR GRANT NUMBER W74V8H-04-P-0485
	5b. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Randolph M. Jones (Soar Technology), Ron Arkin (Georgia Institute of Technology), and Nahid Sidki (Science Applications International Corporation).	5c. PROJECT NUMBER 861
	5d. TASK NUMBER
	5e. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Soar Technology, Inc. 3600 Green Court, Suite 600 Ann Arbor, MI., 48105	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Institute for the Behavioral and Social Sciences 2511 Jefferson Davis Highway Arlington, Virginia 22202-3926	10. MONITOR ACRONYM ARI
	11. MONITOR REPORT NUMBER Contractor Report 2005-03

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Contracting Officer's Representative: Paula Durlach. Subject Matter POC: Randolph M. Jones
This report is published to meet legal and contractual requirements and may not meet ARI's scientific or professional standards for publication.

14. ABSTRACT *(Maximum 200 words)*:

The objective of this work is to design a dynamic intelligent terrain analysis and tactical support system (ITATSS). The system will enable unmanned combat and support vehicles to achieve significant new levels of autonomy, mobility, rapid response, coordination and effectiveness, while simultaneously enriching human–robot interaction, expanding tactical capabilities, and reducing human workload. ITATSS integrates work in intelligent agent architectures for decision support, low-level feature processing, for analyzing terrain and situational features, and robot sensorimotor interfaces. There are currently mature existing tools that handle these capabilities separately, but ITATSS will integrate them into a single architecture. One advantage of such an integrated architecture is that it will help make all of the digital aids familiar and useful to human operators.

This report provides a document to guide the design, development, and evaluation of ITATSS. This should serve as a solid design document for any future efforts to build applications in this area. As part of the design, we have identified a large number of requirements on system components, and any system designed for this application area should meet these requirements.

15. SUBJECT TERMS
Intelligent Terrain Analysis and Tactical Support System (ITATSS)

SECURITY CLASSIFICATION OF			19. LIMITATION OF ABSTRACT	20. NUMBER OF PAGES	21. RESPONSIBLE PERSON
16. REPORT Unclassified	17. ABSTRACT Unclassified	18. THIS PAGE Unclassified	Unclassified		Ellen Kinzer Technical publication Specialist 703-602-8047

Table of Contents

List of Figures

1 Executive summary

1.1 Research requirement

The objective of this work is to design a dynamic intelligent terrain analysis and tactical support system. The system will enable unmanned combat and support vehicles to achieve significant new levels of autonomy, mobility, rapid response, coordination and effectiveness, while simultaneously enriching human–robot interaction, expanding tactical capabilities, and reducing human workload. The effort in Phase I will be to formalize the requirements on the components and interfaces that will make up the Intelligent Terrain Analysis and Tactical Support System (ITATSS) architecture.

1.2 Procedure

For the Phase I effort, we have merged top-down and bottom-up approaches to design an extensible architecture for integrated terrain analysis, decision support, and robot control. Top-down constraints come from our analysis of the architecture's application, including specific requirements analyses for each of the architecture's components. These combine with additional constraints we have imposed on our design to foster extensibility, configurability, and reusability in the architecture. Bottom-up constraints on the architecture's design come from our survey and analysis of a wide range of existing systems that make up candidate implementation systems for the various components that will integrate into ITATSS.

1.3 Findings

This report serves as an initial design document for the Intelligent Terrain Analysis and Tactical Support System (ITATSS) architecture. The goals of this architecture are to:

- Provide integrated presentation of terrain, tactical, situational, and mission information to enhance the situational awareness of a ground robot teleoperator.
- Provide an intelligent user interface for tele-control of unmanned ground vehicles (robots).
- Provide automated decision support to the teleoperator in all phases of the mission (from mission planning through execution and debrief).
- The decision support system will aid in mission rehearsal and training of teleoperators.
- The decision support system will also assist in reducing the teleoperator's cognitive load and enhancing situational awareness during mission execution.
- Provide a component-oriented architecture that can easily "plug in" new technologies as they become appropriate.
- Provide an extensible framework that could ultimately serve as the basis for autonomous ground robot control.

ITATSS integrates work in intelligent agent architectures for decision support, low-level feature processing, for analyzing terrain and situational features, and robot sensorimotor interfaces. There are currently mature existing tools that handle these capabilities separately, but ITATSS will integrate them into a single architecture. One advantage of such an integrated architecture is that it will help make all of the digital aids familiar and useful to human operators. A current obstacle is that there are many separate tools for automation and analysis, and it is difficult for a human operator to master all of them or even to be aware of their individual strengths and weaknesses. We hope to alleviate this situation with an integrated presentation of tools and interfaces that is also assisted by the decision support of an embedded intelligent agent.

It is important to note that this is an initial high-level design document. We have attempted to go into as much detail as is useful, but designs inevitably change when issues arise from initial implementation work and user testing with prototypes. At every design stage, we attempt to include room for testing, evaluation, and adjustment of the design. Thus, this document presents to various levels of detail a description of the different functional components that make up the architecture, as well as what we currently know about candidate solutions and implementations for those components, together with an analysis of how the components must interact.

1.4 Utilization of findings

This report provides a document to guide the design, development, and evaluation of a fairly general but fairly sophisticated framework for unmanned ground vehicle control, integrated with automated decision support and intelligent event and terrain analysis. This should serve as a solid design document for any future efforts to build applications in this area. As part of the design, we have identified a large number of requirements on system components, and any system designed for this application area should meet these requirements. Thus, even if the ITATSS architecture itself is never instantiated, it provides context, requirements, and evaluation metrics for similar systems. The architecture also provides a coherent framework for incremental development of practical applied systems. The overall ITATSS architecture is an ambitious design that will keep down long-term costs in the development of future robot control systems. However, it also informs the design of incremental improvements in existing technology. We will use the ITATSS design to constrain new extensions to existing systems for robot control, terrain analysis, decision support, and digitally assisted situational awareness.

1.5 Report roadmap

We begin this report describing the fundamental design and evaluation principles behind our approach to architecture design (Section 2). We then move to an overview of the architecture (Section 3), highlighting its structure and the advantages of our approach to integrating a variety of functional capabilities and interfaces. The most important components from the user perspective are the user interfaces that provide a variety of tools for maintaining situational awareness before and during the robot mission (Section 4). But we also provide high levels of details on the functional components "under the hood" of the architecture, focusing on a number of existing technologies that should support cost-effective integration into a novel new system (Section 5). We end with a discussion of how the proposed architecture will interact with modeling and simulation environments for training (Section 6), in addition to being deployable for real-world missions, followed by a brief summary and overview of our conclusions (Section 7).

2 Design principles and evaluation

2.1 Design philosophy

The following principles have guided our design process for ITATSS, and will continue to frame the further design and implementation work we perform during subsequent phases of this project. To begin with, we are committed to creating an architecture that is composed of well-defined interacting components that communicate with each other via formal abstract interfaces. Communication between architectural components will occur through a shared knowledge repository (SKR) that maintains a representation of the system's overall situational information, providing appropriate data and context to each of the functional components (see Figure 1).

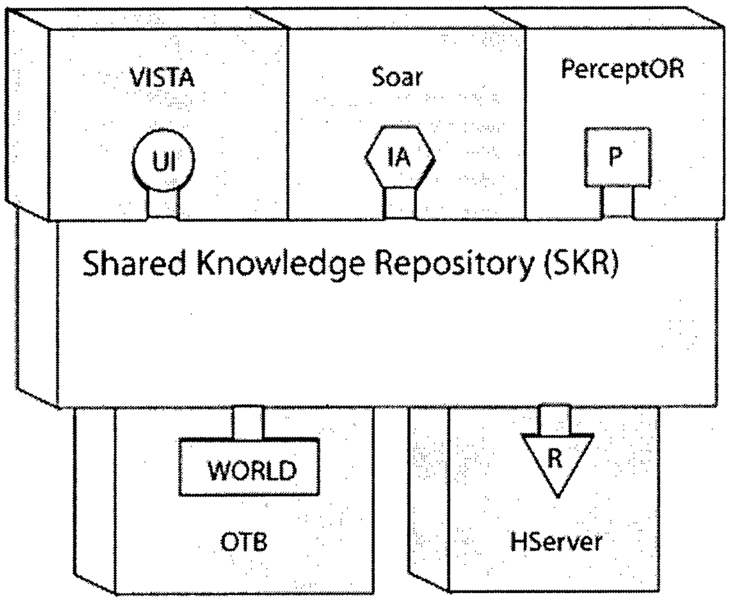


Figure 1. An example instantiation of the ITATSS architecture. The Shared Knowldge Repository provides the system's overall representation of situation, providing context and data to each functional component. Well defined abstract interfaces allow selection of appropriate alternative implementations for the component functions.

The reusable component-oriented design of the architecture will foster future adaptations of the architecture to changing requirements and technologies. It will also enable one of our additional design commitments; to adapt and reuse existing state-of-the-art systems as much as possible through a "synergistic integration" into the new architecture. The ITATSS design and development team has created and is intimately familiar with a number of mature existing software and hardware systems that provide many of the subsystem functions required by ITATSS. With the architecture's design providing well-defined interfaces to each of these types of subsystems, we will have maximal flexibility in terms of selecting from existing technology and Commercial Off-The-Shelf (COTS) solutions, to integrate them as appropriate into the ultimate implementation of the ITATSS system.

An additional overriding part of our design philosophy is to use user-oriented design principles throughout the design and implementation process. Ultimately, the ITATSS architecture will provide a number of robot teleoperator user interfaces that must maximize the effectiveness of training and mission performance. It is imperative to design these user interfaces, and the functions that support them, with the needs and best interests of the user at heart. Thus, we plan to use subject-matter experts (SMEs) throughout the design process, and to involve representatives of the potential user community during the prototype and testing phases of the architecture's development.

Finally, we take seriously the issue of system evaluation from multiple perspectives. We must evaluate the individual components that will eventually instantiate the architecture. We must evaluate the performance and effectiveness of the architecture as a whole, in particular with respect to the usability of the user interfaces. And we must evaluate the training and performance effectiveness of the resulting decision support and awareness system. We will detail metrics and methods for each of these types of evaluation.

2.2 Motivating scenario

From the onset of our project, we have been considering the application of this research in the context of urban warfare, motivated to a high degree by our previous experiences and successes in operating in the McKenna MOUT facility at Ft. Benning, Georgia. Following, we describe a scenario that we envision as one typical and plausible test case for this technology. It is important to remember that the design of our approach is highly generalizable, however, and can be readily expanded to teams of UGVs and UAVs operating in a broad range of terrain types and missions.

> *From a point of departure located several kilometers from the McKenna site a C2 vehicle (e.g., the Georgia Tech Hummer as seen in Figure 16) intelligence arrives indicating the potential presence of a high-value target located in a particular building at the site. A surreptitious mission must be created using a wooded approach area that incorporates knowledge of the terrain for a stealthy approach and entry to the suspected target location. ITATSS is used to incorporate terrain knowledge into the pre-mission plan generation and specification activities on site in support of this mission.*
>
> *After mission specification is completed the robot is operated under a combination of teleoperated, semi-autonomous, and autonomous modes as required, to navigate its way through a complex route involving terrain following behavior to provide appropriate cover for its approach, Continual real-time feedback is available to the operator of the mission during its execution.*
>
> *Depending on the particular capabilities of the operator, the generated mission profile, and the current CONOPS, the decision support agent (DSA) provides a series of suitable user interfaces that support mission completion, that dynamically change during the mission and that provide a markedly reduced cognitive load on the operator as the mission progresses. The operator is kept continually aware of any deviations from expectations regarding terrain while ITATSS concurrently updates the terrain database for future operations. The actual mission progresses through a series of legs including open-terrain crossing, dense woods navigation, village approach, observation post occupation, and target recognition, confirmation, and neutralization.*

2.3 Metrics and evaluation

As we move further into the realm of intelligent unmanned systems, the problem of devising performance metrics for assessing system intelligence looms larger and larger in importance. This is an issue the field of AI, robotics, and intelligent systems has ignored for the most part. Most research results are in the form of demonstrations rather than experiments with data that is quantitative and referenced against ground truth. There are a few benchmarks or standardized tests wherein performance can be compared. For example, perhaps the most common metric for human intelligence is the intelligence quotient (I.Q.). However, there is great controversy in what I.Q. is and how it should be measured. There are, of course, many mental skills and abilities that can be measured. These include the abilities to read and write, to calculate with numbers, to reason with logic, to remember what was seen and heard, to perceive patterns, to understand relationships, and perform geometrical transformations. In contrast to previous work, performance metrics and systems evaluation are a centerpiece of our work with the ITATSS architecture. We will use metrics throughout the design process to identify quantitatively the best architectural components and integration.

The success of any program is measured by its ability to perform the tasks that were intended. This in turn can be subdivided into specific capabilities, some of which may be interdependent. For ITATSS, the success will be measured by the success of decision support agent (DSA) that incorporates valid understanding of the operator task. The success of the individual DSA capabilities will depend on the performance of the underlying components that provide that capability. Mostly, success is not a binary state; there are infinite possibilities between what one might classify as complete success or complete failure. The component level performance, in most cases, can easily be measured directly, or established indirectly. Our team proposes to implement a method to aggregate performance metrics from components level all the way to the entire ITATTS using a novel schema involving the Ordered-Weighted Averages (OWA). The OWA method facilitates the aggregation of different types of metrics into a single value based on fundamental assumption of the importance of each of the component performances. Similar metrics can be combined arithmetically to gauge combined performance; but dissimilar metrics cannot be combined easily. For example when two robotic vehicles are deployed, the total area of coverage can be calculated from the individual coverages, with the knowledge of the amount of overlap. However, if efficiency is defined as a function of the speed of deployment and the number of targets acquired, it is hard to roll speed (with units of length/time) and a simple number of targets (a unitless quantity) into a simple resultant metric in an arithmetic fashion.

The OWA methodology combines the low-level metrics by determining weights for each of them based on a fuzzy-logic approach. One can specify the linguistic qualifiers that are used to sort and combine the metrics. An example of a qualifier will be "*most of the metrics/attributes are satisfied*". Also the relative importances of each of the components to the overall ITATTS system can be specified. This internal sorting of the metrics will allow the processing of scenario-based system metric. For example in one scenario, rapid response might have paramount importance while in another coverage of the scene may outweigh rapid response in order to avoid surprises. Thus the same system will score differently under different scenarios. The desire, however, is to maximize the performance in any situation.

ITATTS-wide metrics desired by user are 1) Autonomous Coverage, 2) Rapid Response, 3) New Platform Integration, and 4) Prioritization of user requests. These metrics will be carried down to the sub-systems and component-level metrics and will be classified as contributing to one or more of these system metrics. For example each UGV will have a theoretical coverage area, which is modified in actuality by parameters such as obstructions, sensor performance etc. Total coverage can be determined by the union of the individual coverages (this accounts for the coverage overlap). However, even though the coverage overlap reduces the total coverage, it does increase system robustness by providing a level of redundancy.

2.3.1 Metrics for decision support agent development

The program progress will be assessed internally using a combination of subsystem-level metrics as well as requirements-driven metrics. The subsystem-level metrics are aggregated from component level measureables. The success of the system integration of the decision support architecture effort will be put in the context of achieving the capabilities specified.

The appropriate metrics will also be simulated in the simulation environment. Thus the performance of the simulated components can be directly correlated to the actual decision support system performance, as shown in Figure 2.

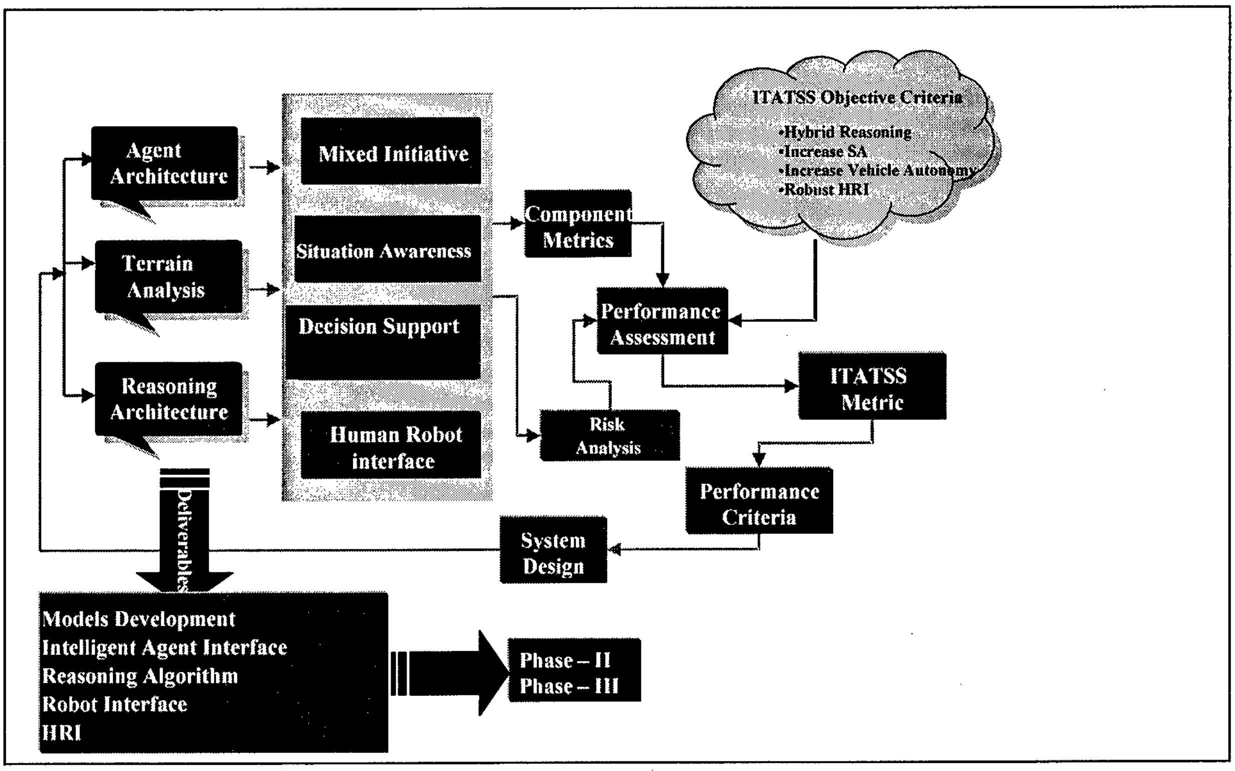


Figure 2. The Proposed ITATSS system analysis process uses agent architecture, terrain analysis, and reasoning architecture.

2.3.2 Relationships decision support agent and system level metrics

As the requirements of the decision support system and its components are derived from the overall program requirements, these metrics can be aggregated to provide program-wide metrics. Basically the component and subsystem level metrics will be classified according to the role they play in the four main program level metrics:

1. Intelligent agent interface
2. Human interface
3. Robot interface
4. Low level feature processing

Thus the lower level metrics can be aggregated using OWA to provide the four program level metrics. Note that not all of the system/component level metrics will contribute to the program level metrics. Also it is important to note that each of the internal metrics need not have the same "importance" when the program-level metrics are computed using the OWA method as shown in Figure 3.

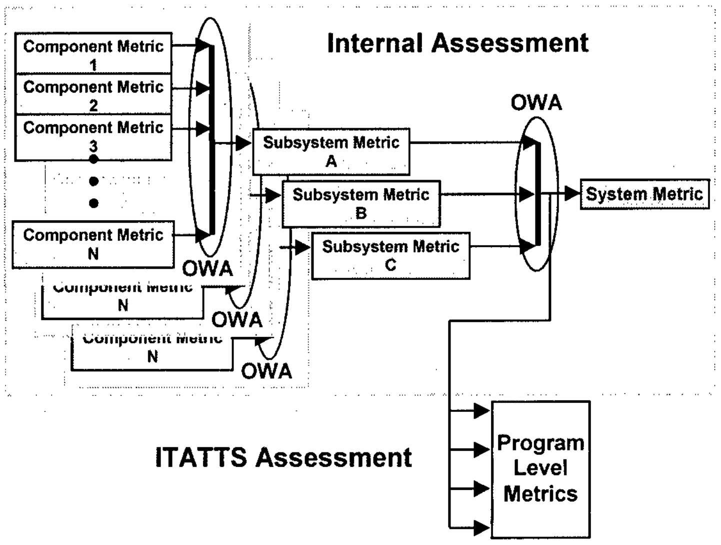


Figure 3. Aggregation of the component-level metrics to sub-system level and then to platform level enables the evaluation of the progress internally.

2.3.3 Metrics for integration and experimentation

The metrics can be classified according to their relationship with the primary technical objectives – 1) Decision support agent, 2) user interface and robot control, 3) robotic interface and 4) System Integration. In each of the categories, the requirements drive the choice of metrics.

2.3.3.1 Decision support agent

The decision support agent will take data and extract from it an understanding of the best actions to take next. This requires a process of assimilating the data to produce information, situation awareness and finally optimal response. These levels of assimilation relate to the level of decision making that is needed (e.g., UGVs with different sensor modalities generate so much data that is it hard to make sense of it). The decision support should be measured according to the following criteria.

- **Usability:** Usability will be measured in part in terms of the time taken to execute a complete command. There will also be other traditional usability metrics measuring ease of use, ease of learning, and system effectiveness and efficiency. Utilities to scan the system logs will be developed to provide this information.
- **Ease-of-Use:** Ease of use can be measured by the number of commands issued to generate a particular system response.
- **Dimensionality:** Dimensionality is defined as the number of requests that the system can effectively respond to in unit time.
- **Robustness:** Robustness is defined as the degree to which mission effectiveness is maintained with changes in usage. This metric reflects the stability of the system that can be measured by the rate of change of the aggregated performance metric at that level – component, sub-system, or system.
- **Responsiveness:** Responsiveness refers to the capability of the system to return a response to a user command in a reasonable period of time. When more users exercise the system it will slow down. Hence responsiveness can be defined by the ratio of the response time under normal load to that when only one user is present.
- **Mission Effectiveness:** This measures the correlation between the actual mission effect and the expected mission effect. Even though this might be a subjective concept, it can

be quantified on a preset scale. The mission commander sets this metric based on the preset scale.

The decision support agent, user interface, and robot control highlight the numeric performance goals for the ITATSS system, but we will also collect and analyze experimental data to assess the compliance and learning challenges of the program. We will develop reference scenarios; simulated agent and operational situations; procedures for scoring compliance; and methods for quantifying and characterizing a commander's or user's interaction with the ITATSS system.

2.3.3.2 Robot and sensor modeling metrics

Sensor simulation will allow the evaluation of more advanced behavioral algorithms, including the fusion of several perceptual processes that will provide a more accurate and meaningful scene characterization process that will support a greater level of useful autonomy. The metrics can be classified according to their relationship with the primary technical objectives:

1. Expressivity
2. Extensibility
3. Efficiency
4. Bandwidth and
5. Translation

In each of the categories, the requirements drive the choice of metrics.

- **Expressivity:** Expressivity is defined as the ability of a model to simulate the behavior of a platform or a sensor. Even though it can be measured as the number of behaviors appropriately simulated, one can also include the level of fidelity of simulation of each simulation as a fraction between 0 and 1.
- **Extensibility:** This metric is defined as the speed with which a new concept or platform behavior can be included in the simulation system. For faster integration of new concepts, the underlying simulator architecture has to be flexible, modular, and yet most of the parameter space has to be accessible to all modules.
- **Efficiency:** Efficiency is determined by the speed of simulation of the behaviors. This implies that the level of complexity has to be commensurate with the hardware available. The use of parameterizations to represent physical processes will increase efficiency. The behaviors that are not resolvable in a given scenario can be switched off or parameterized to provide the effects into the overall picture.
- **Model Library:** The length of time it takes to write library modules that simulate a behavior will dictate how many behaviors and platforms can be included in the library. A key to the successful creation of compact libraries is the abstraction of basic behaviors as library functions that are then called by the library modules that make up the model of a platform or sensor.
- **Translator:** The translator refers to the component that takes the individual steps in a system plan and translates them to the commands of the specific platform simulator.

Accurate representation of terrain features in sensor modeling will support the level of ambiguity that is found when operating in the real world. Also accurate simulation of such conditions can compress the development time for the ITATSS system and even determine feasibility before a significant program commitment is made.

2.3.3.3 Mission planning and control metrics

Key innovative features of our approach include using an advisable planner to implement commander's guidance and allowing humans to specify advice and partial plans at multiple levels of abstraction. Another innovative feature is using the concept of *value of information* (VOI) to distinguish important alerts and deviations during execution. VOI refers to the pragmatic import the information has in making informed decisions. We will define a set of plan-aware, situation-aware *policies* by interacting with subject matter experts (SMEs) and encoding knowledge about desired responses to execution events. The commander can select the desired policies and preferences to be active for a particular mission plan or operation. Policies can also specify a user-specified parameter space that defines situations of special interest and specifies instructions for how to respond. For example, a policy might require UGVs to divert to secondary targets if visibility is below a certain threshold. We encode policies in machine-understandable procedural representations, making our system responses flexible and sensitive to changes in context, requirements, and policy. We will use performance metric to quantify the advisable planner for the following criteria:

- Command Latency
- Information Latency
- Robustness
- Adaptability
- Utilization

A multi-step approach to metrics collection and analysis may be done to improve the concept of *value of information* and their tactical implications. Figure 4 demonstrates a spiral-based approach to examining mission planning under simulated and live conditions to further the experimentation process without the need for large-scale assets.

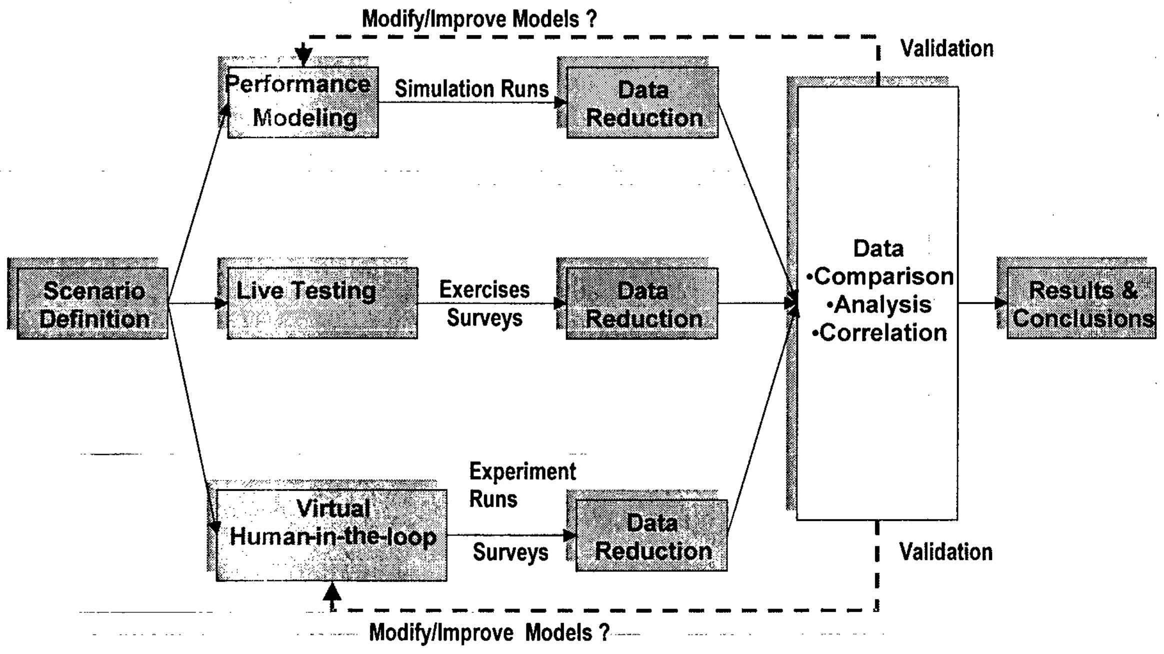


Figure 4. Performance metrics collection methodology.

2.3.3.4 Usability testing

Our group has extensive experience in performing formal usability evaluation studies for both run-time and permission operator interfaces for controlling autonomous systems, dating back to the mid-1990s. These results have been applied within many different DARPA programs including DARPA MARS, TMR and UGV Demo II. It is highly desirable to apply these formal evaluation methods, both summative and formative, to ITATSS. It is recognized, however, that much of this work may need to be delayed until Phase III of this program due to the extensive labor and associated costs involved in testing dozens of users subjects scientifically. Prototype tests however can be constructed in support of this work in Phase 2, accompanied with reliance on the evaluations already in existence from the numerous user interfaces already developed and subjected to formal usability studies residing within the MissionLab system components that are relevant for reuse within ITATSS.

3 ITATSS architecture overview

From a general perspective, the ITATSS system supports three primary entities: the operator, the robotic vehicle, and the decision support agent. The operator teleoperates the robotic vehicle with support from the decision support agent. It is crucial that each of these components can get and provide the appropriate knowledge and information to each of the other parts.

The primary organizational concept behind ITATSS is a shared knowledge repository (SKR). With this core repository of information, there will be an engine to maintain, control and distribute the shared knowledge and information that the system entities must pass among each other. The current design is intentionally not committed to a particular implementation of the SKR. It could be implemented on a central server or in a distributed database, depending on the application platform. In any event, the SKR and the engine that serves it are shared by all of the external components. The components communicate with each other in a "blackboard" fashion, by depositing structured knowledge representations when they have something to contribute, and monitoring the repository for relevant information and directives. This provides a flexible and effective way of integrating each of the components using a neutral style of knowledge representation. Any new components to the system will be able to integrate with this store of knowledge in the same way as the original components.

Each component of the system includes subcomponents that allow the component to interact with the others through the SKR. For the human operator, these subcomponents consist of the various user interfaces described in Section 4. But for the DSA and the robot, these subcomponents are automated software systems. Figure 5 displays the communication interdependencies between architectural components (which in the ultimate implementation will all happen by communication through the SKR).

External information to the architecture comes in a variety of forms, each describing some aspect of the overall situation. For our purposes, there are five different types of external information that make up the system's overall situational awareness:

1. Events (generated by the situation event detector)
2. Intelligence (generated by external intelligence gathering and communicated by existing digital systems)
3. Terrain (generated by the terrain feature analyzer as it updates the dynamic terrain database)
4. Mission and operations order (generated by external mission briefing systems), and
5. User profile (generated by ITATSS configuration software)

ITATSS builds structured representations of each of these types of information, and integrates them in the SKR. Subsets of the SKR are then monitored by each system component in order to contextualize component behavior. For example, relevant elements of the SKR will be displayed to the user operator via the ITATSS user interfaces. Almost all of the elements of the SKR will provide the working blackboard memory for the decision support agent, so it can make all of its decisions, recommendations, and explanations in the appropriate context of the current situation. The decision support agent will also use operator preferences in the user profile to make recommendations about which tools and user interfaces to display on the Operator Control Unit at any given time. The robot systems will monitor representations of executable robot plans in the SKR, and execute them when directed to by the operator or the decision support agent.

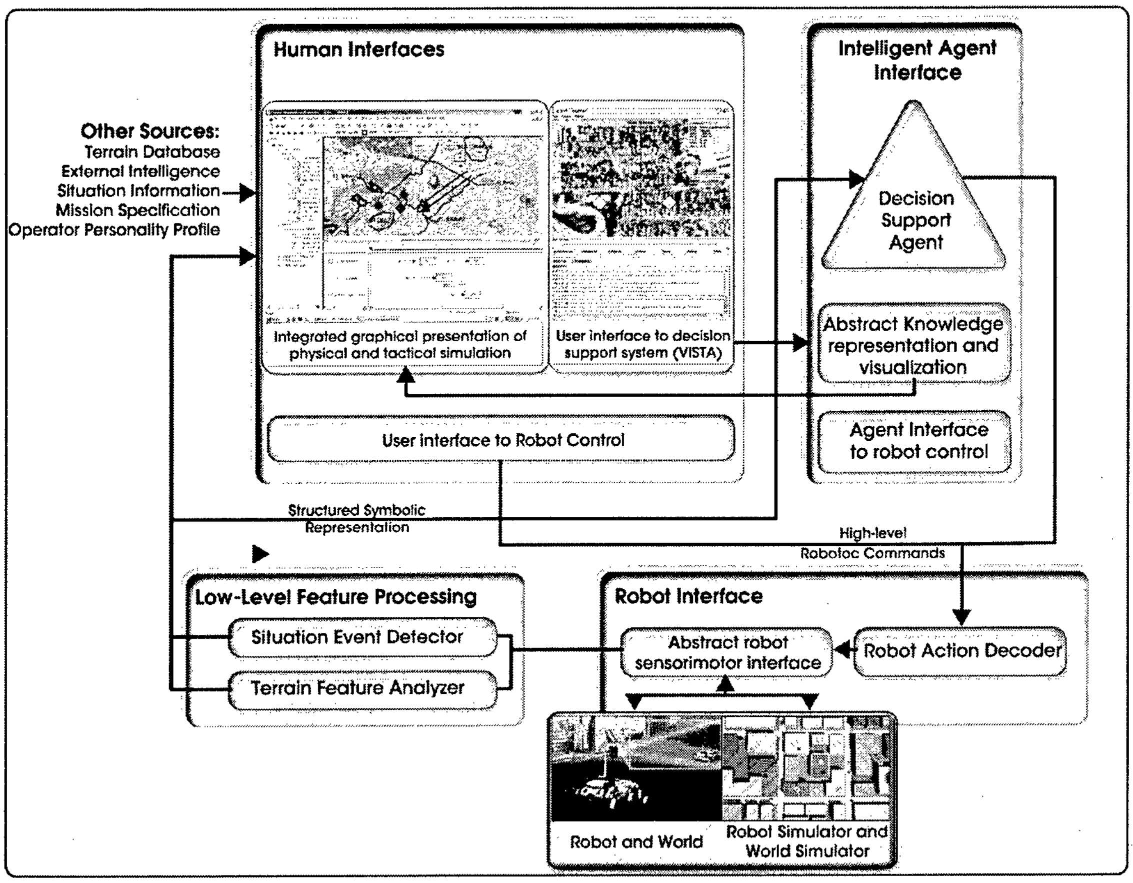

Other Sources:
Terrain Database
External Intelligence
Situation Information
Mission Specification
Operator Personality Profile
Human Interfaces
Integrated graphical presentation of physical and tactical simulation
User interface to decision support system (VISTA)
User interface to Robot Control
Intelligent Agent Interface
Decision Support Agent
Abstract Knowledge representation and visualization
Agent Interface to robot control
Structured Symbolic Representation
High-level Robotoc Commands
Low-Level Feature Processing
Situation Event Detector
Terrain Feature Analyzer
Robot Interface
Abstract robot sensorimotor interface
Robot Action Decoder
Robot and World
Robot Simulator and World Simulator

Figure 5. Notional depiction of the ITATSS architecture.

One of the primary technical innovations of the ITATSS design is that it will be designed from the ground up as a component-oriented composable and extensible architecture. The SKR provides a generic formal programmer interface to guide the communications between components. As long as any component provides the proper functionality and integrates with the SKR's programmer interface, it will be able to plug in to provide services to the architecture (see Figure 6). This component-oriented framework will allow us to explore and identify a variety of options for each functional component within the architecture. It will also allow us easily to extend and adapt the architecture as changes in technology warrant. Essentially, the well-defined interfaces to the SKR allow ITATSS to provide a palette of choices for systems designers, users, and the decision support agent to select from for various applications and instantiations of the architecture.

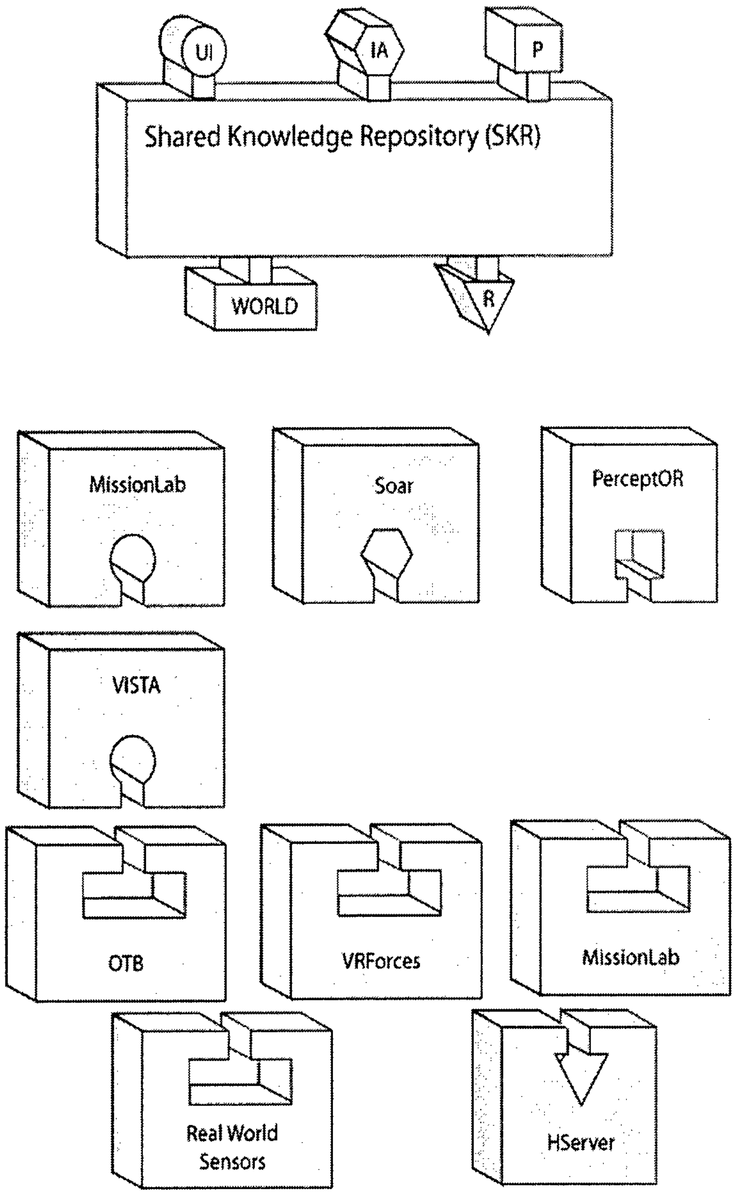


Figure 6. The Shared Knowledge Repository (SKR) provides well-defined abstract interfaces allowing a pallette of instatiation options for the various functional components.

One key component of the overall architecture is the decision support agent. This agent consists conceptually of two parts; an intelligent agent architecture and a knowledge-based system implemented within the programming language provided by the architecture. We will discuss requirements on the best choice of intelligent agent architecture in Section 5.1.1. We will also describe in detail the design process for the decision support agent in Section 5.1.2.3. We must also describe the various requirements on the behavior that the decision support agent must generate.

In order to provide useful decision support, the decision support agent must first have a competent knowledge base for performing the task. This knowledge base defines the high-level interpretations, goals, and procedures that an expert reasons about when performing the task. With the help of a subject matter expert, LTC Scotty Abbott, USA (ret), we performed an initial analysis of high-level capabilities involved military ground robotics task domains. We will describe these capability requirements in various sections detailing decision support involving the various functional components and user interfaces in ITATSS. Our discussions will address the knowledge requirements both for *task performance* capabilities as well as *decision support* capabilities. Our experience with knowledge-intensive intelligent agents dictates a three-pronged high-level organization for all of this knowledge in the decision support agent:

- situation interpretation,
- purposeful reasoning and behavior, and
- external action.

Analysis of the task and of experts who know how to perform the task reveals the representation terms necessary for understanding, the interrelated goal structures necessary for providing context and direction to interpretations and actions, and the complete set of external behaviors that provide the possible space of observable actions involved in the task. In following subsections we further discuss each of these three types of knowledge in the context of various functional components and interfaces.

4 User interfaces

From a functional and user perspective, one of the most important aspects of ITATSS is the set of user interfaces. ITATSS must provide appropriate and user-friendly interfaces for each phase of the teleoperator's mission (as well as mission planning and possibly also debriefing). It is imperative to follow appropriate practices for user-centered design in order to maximize the effectiveness of the overall system. Building good user interfaces requires a combination of functional analysis for each task plus interaction with potential users of the system to determine how they can best interact with the system. This section describes the user-oriented approach we plan to use for interface design, and then provides more detail on the various interfaces that the user will interact with during different phases of the mission. In addition, through all phases of the mission, the user will also interact with the decision support agent.

User-centered design demands that we involve potential users and subject matter experts from the early design stages throughout the design an implementation of the system. Thus, we have already had initial discussions with a subject matter expert for military robotic teleoperation. Perhaps the most important area where subject matter expertise is necessary is in the design of the operator's graphical user interface to the ITATSS system.

For this project, we also have the advantage of building on a number of existing systems that already have incorporated user-centered interface elements. The graphical user interfaces for the ITATSS system will provide the user with the most appropriate levels of interaction with several subsystems, including the systems for:

- Analyzing terrain and map information
- Analyzing integrated mission and intel information
- Planning missions
- Rehearsing missions
- Executing missions
- Interacting with robot platforms
- Interacting with the decision support agent, including run-time support and training support

Requirements for the design of the graphical user interfaces must maximize the usability and usefulness of the software interfaces to each of these subsystems. Many of these systems relate to existing systems that have well-designed and tested user interfaces. For example, the OTB, Demo II and Perceptor systems provide mature tools for terrain analysis, and also for integrating (to some extent) intelligence information with the terrain analysis systems. The MissionLab system provides tools for planning and rehearsing missions, and also provides a platform for mission execution through teleoperation of various robot platforms.

Interface design for interacting with the decision support agent will closely follow the behavior requirements for the decision support agent. The decision support agent essentially must have knowledge of two broad domains: the mission tasks that the operator will be executing and how best to interact with and support the operator in mission planning, execution, and training. Elsewhere, we describe elements of an initial cognitive task analysis collected from our subject-matter expert, which will be used to drive the performance and interaction of the decision support agent. Aside from the task requirements, the overriding philosophy behind the support interface will be to interact in as natural a way as possible with the operator. We envision the decision support agent to be a synthetic replacement for a human advisor who might otherwise be supporting the operator. Thus, the interface will provide multi-modal presentation of information (natural language descriptions together with highlighted graphical displays), and will allow interactive explanation of decisions through template-driven messages. The infrastructure for this interface will be provided by the VISTA framework (Taylor et al., 2002), or a suitable substitute. VISTA provides architecture-neutral visualization, presentation, and explanation facilities for intelligent agent systems.

Displays developed within VISTA include graphical elements representing the knowledge and belief structures that make up an intelligent system's representation of situational awareness. Because VISTA represents these knowledge structures in an agent-independent manner, it follows the same component-oriented principles as the ITATSS architecture, making it relatively easy to integrate with other systems. The ITATSS user interface will overlay VISTA's verbal and visual representations of the decision support agent's knowledge with the other situational information that is present within the SKR. Thus, the ITATTS graphical displays will merge "ground truth" information from external sources with "interpreted beliefs" generated by the decision support agent, as well as symbolic interpretations input by the human operator.

For the input interface, the human operator will have separate modes or windows for interacting with the physical robots, external information sources, and the decision support agent. Interaction with the decision support agent will again occur through a VISTA-like framework, passing messages and questions from the operator. Thus, the decision support agent's behavior will be driven both by its observations and interpretations of the mission situation and direct interaction with the operator.

The remainder of this section describes the various ITATSS user interface elements for all of the phases of UGV teleoperation missions.

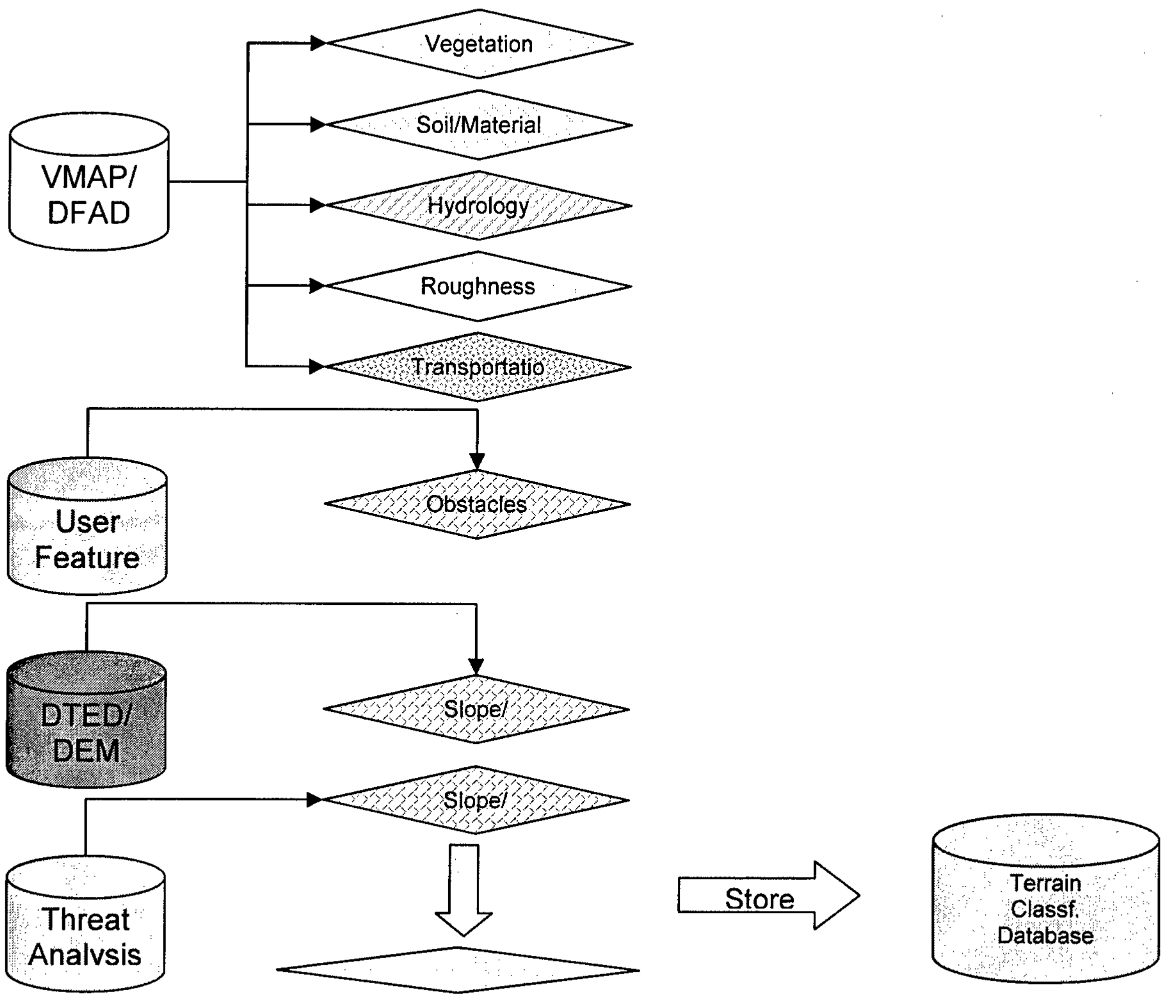


Figure 7. Elements of the static terrain classification map.

4.1 Pre-Mission situational awareness

The first phase of teleoperator interaction is mission planning and rehearsal. The operator must develop appropriate awareness of the tactical situation and the mission, select appropriate tools and interfaces for mission planning, as well as map and terrain analysis, and plan the mission and various contingencies as flexibly as possible. The decision support agent and ITATSS user interfaces must support all of these tasks, particularly by providing the user with useful representations of situational awareness for the pre-mission situation. The following subsections details user interfaces for relevant components of pre-mission situational awareness.

4.1.1 Terrain analysis

In typical state-of-the-art systems for semi-autonomous cross-country navigation, routes are planned as a sequence of waypoints defined in some global coordinate system. The robot uses a pursuit algorithm to steer towards the next waypoint along the route using GPS and/or inertial navigation to estimate the vehicle's position in global coordinates without any use of features in the local environment. As a result, when there are large errors in vehicle position estimation, it frequently leads to erroneous behavior.

This is in contrast to how Soldiers are trained to do navigation. They begin at the start point and use a compass to face in the proper direction, then sight in on a landmark that is located on the correct azimuth to be followed. Next, close the compass and proceed to that landmark. Finally, repeat the process as many times as necessary to complete the straight-line segment of the route. The landmarks selected for this purpose may be uniquely shaped trees, rocks, hilltops, posts, towers, buildings, or anything that can be easily identified and display in the OCU. In robotic terms this corresponds to navigating straight segments between waypoints using visual navigation to a sequence of distinguishable visible features along the azimuth.

Terrain classification and analysis embodies the world model for any given level within the UGV mission planning and control system. The terrain classification module is primarily used to initialize and maintain the world model for the UGV (see Figure 7). We will develop realistic behavioral representations of the terrain for the user interface. Below is an example of the OCU that we developed for Demo III program.

UGV outdoor research can be divided into three coarse areas of focus: depot/dense urban, on road, and off road navigation. While significant progress has been made in these areas they are still treated as separate problems. In a typical example deployment scenario the vehicle is turned on and driven by joystick out of a high bay, it is stopped at either a road or trail deployment site, software for off-road or on-road navigation is started, and the vehicle is set loose.

We propose to unify these traditionally separate areas of research under the ITATTS program to allow seamless terrain analysis and navigation from depot to mission completeness.

The Demo III Operator Control Unit user interface consists of a full-screen 2D map-based application with military and robotic symbolic overlays to annotate the mission goals and objectives. The key man-machine interface design considerations for the Demo III OCU were derived from the Demo III mission planning and control requirements, the Department of Defense Human Computer Interaction Style Guide, the Joint Technical Architecture-Army Weapon Systems Style Guide. Additional US Army documents influenced the OCU design ensuring OCU usability for future military field operations.

The OCU screen-layout consists of four major components: the large menu buttons area, the tool pallet area, the primary map display area, and the status area denoted in Figure 8. The position of these major components can be reconfigured to provide preferential left or right-handed operations.

The menu buttons area contains large menu buttons for easy touch-screen access to mission critical operations while on the move These menu buttons provide access to the majority of the convenience tool dialogs containing all of the map management, mission planning and plan execution and vehicle safety functions.

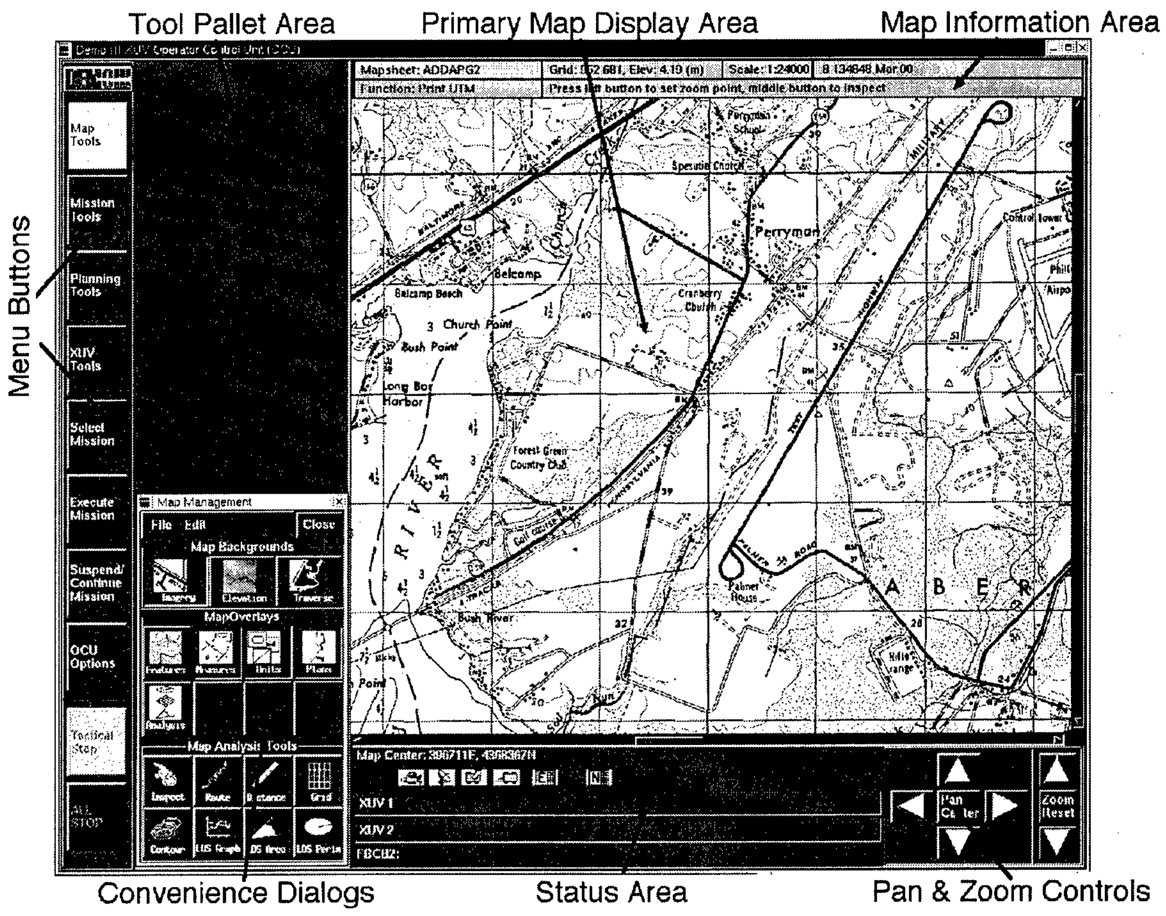


Figure 8. OCU display layout.

The tool pallet area reserves screen space for the convenience dialogs to minimize obscuring the primary map display area. Each of the dialogs provides a set of tool buttons for manipulating the background map and graphical symbology of the primary map display area. These tools allow the operator to specify, edit, move, and delete various military and robotic graphical symbols interactively on the background map, denoting specific mission or robotic vehicle actions to be performed during the scout missions.

The primary map display area is the main focus of the OCU. All of the OCU convenience dialogs and menu buttons are used to manipulate elements on the map view. It consists of three components: the map information area, the map display area, and the map pan and zoom controls. The map information area provides textual map information about the current map product and scale being displayed. Additionally, the current cursor map coordinates and input mode instructions are presented to provide operator feedback when manipulating graphical overlay elements and iconology. The map display area provides the ability to display color ortho-rectified map imagery, elevation or traversibility renderings as a background with layers of graphical overlays for various mission planning and map analysis functions. This view is scrollable and scalable to provide mission specification, execution and monitoring of the UGVs at the appropriate level of detail to ensure situational awareness of the mission. Each graphical overlay layer can be hidden or shown from the map management convenience dialog shown in Figure 8 to de-clutter the map display. Elements in the graphical overlays can be selected for repositioning, editing or deletion as required directly from context sensitive menus on the map display. During mission execution, the UGVs and their respective robotic plans are updated to reflect the current position and activities of the vehicles. These individual vehicles can be tasked directly from context sensitive menus to suspend or redirect their current activities. The map pan and zoom controls provide large touch-screen accessible controls to adjust the currently viewed portion and scale of the map and overlay graphics.

The status area provides textual feedback about the health, status and actions the UGVs are executing. This includes the UTM Grid position of the vehicles, the current speed and direction of travel, and any message from the vehicle regarding the current actions being performed such as executing Remote Sensor Target Acquisition (RSTA) search, executing move-on-route, etc.

All of the components work together seamlessly to provide consistent and intuitive tasking, control and monitoring of the multi-vehicle UGV teams performing cooperative scout skills and missions.

4.1.2 Integrated situational information

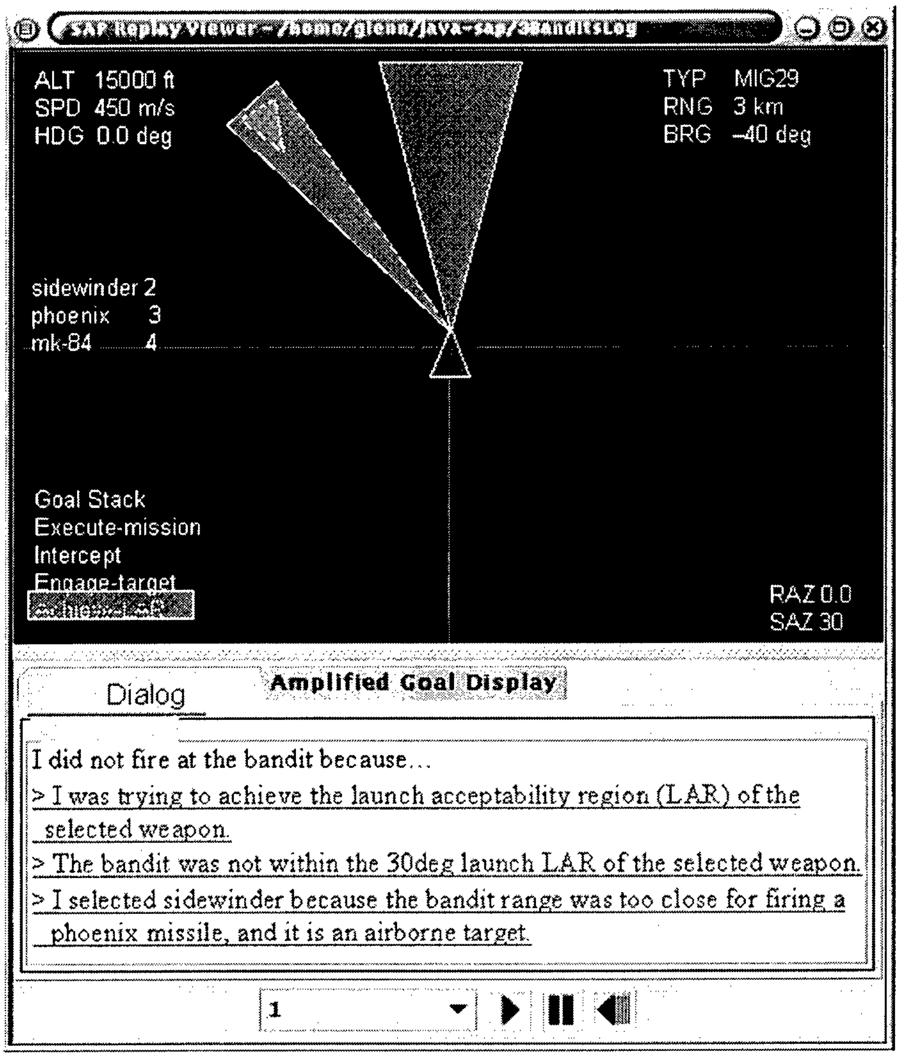
The previous section describes systems that perform a "pure" form of terrain analysis, in which the analysis can be considered mostly as a data-driven process. Terrain analysis systems encode a number of primitive terrain features that they attempt to detect. Then, purely from sensed data and the terrain map, these systems classify new perceptions into this feature set and merge them into the dynamic terrain database. For the most part, such systems do not process the terrain data in the context of higher-level situational constraints, such as specific goals and aspects of the mission, interface preferences expressed by the teleoperator, or situational intelligence information provided by external information systems.

One emphasis of the design task for ITATSS will be to develop the notion of "augmented terrain analysis", which introduces situational context into the interpretation of terrain information and other, newly detected situational events. One aspect of this point of emphasis will be addressed by the knowledge representations we develop for the shared knowledge repository. We will specify a knowledge-based situation-representation language that uniformly integrates all the types of situational information that are relevant to completing the mission, including detected events (generated by the situation event detector), dynamically gathered intelligence (generated by external intelligence gathering and communicated by existing digital systems), terrain features (generated using traditional data-driven terrain feature analyzers), mission constraints and goals (generated by external mission briefing systems), and user preference information (generated by a the user-profile editor in the ITATSS configuration software).

The other primary emphasis of augmented terrain analysis will be on actually using the contextual situation information to influence the normal terrain analysis process. Lower levels of this contextualization can take place by modifications to the existing systems that perform traditional terrain analysis. However, the primary activities along these lines will be implemented in the knowledge-based decision support agent, described in the next section (4.1.3).

4.1.3 Decision support for situational awareness

A major part of the decision support agent's task will be to process information that lies on the continuum of "augmented terrain analysis" and "augmented decision making". The agent will use the integrated situation information to guide and focus new interpretations of the situation, including the processing of new terrain feature data and new detected events. The agent will also use this information to augment the representation of situational awareness (held in the SKR) that will ultimately be displayed and highlighted to the user, to augment the user's decision-making processes. Figure 9 show an example Situational Awareness Panel display for a knowledge-intensive intelligent agent that flies a number of military missions for simulated fixed-wing aircraft. Rather than displaying ground truth, this display presents graphical visualizations of the agent's beliefs and interpretations of the situation. Current research and development is creating additional integrated functionality for present verbal and graphical explanations of the agent's decision process, on request from the user. We will develop similar explanatory aids for the DSA.



I did not fire at the bandit because...

> I was trying to achieve the launch acceptability region (LAR) of the selected weapon.

> The bandit was not within the 30deg launch LAR of the selected weapon.

> I selected sidewinder because the bandit range was too close for firing a phoenix missile, and it is an airborne target.

Figure 9. VISTA display of an intelligent agent's situational awareness, augmented with decision explanations.

4.2 Mission planning and specification

Prior to the execution of a mission it is assumed that a CONOPS will be provided to the operator to serve as the basis for its creation. A CONOPS is designed to provide a high level description of how a mission or assigned task is to be conducted. CONOPS are often developed for newly assigned missions for which there is no previous experience or for missions that are affected by the emergence of a new technology. A CONOPS defines and clarifies the purpose, scope, intent, activities, and responsibilities necessary to initiate, execute, and sustain a mission or task. The purpose of the CONOPS is to ensure common understanding and unity of effort for all organization (behaviors) members that are required to support the mission/task.

Mission behavior involving commonly needed functions and simplified user interfaces can be programmed in advance so that warriors can use them without explicit mission planning or mission control. As each behavior is added, user workload is reduced for the task it was built to perform. This is analogous to developing a hard-wired machine to perform a specific factory task. As long as a process is routine and predictable, engineers can automate it. But war is not routine or predictable. The enemy will always find a way to adapt to yesterday's automated tactics.

Traditional mission planning and mission control techniques may be labor intensive, but at least they let the warrior express creativity and adapt to changing conditions without calling home for a new engineering solution. We need to combine the power of behaviors with the flexibility of manual planning.

The general solution should be a flexible set of simple behavior building blocks and interface elements that can be assembled in the field to define custom Mission Behaviors on the fly. This is analogous to upgrading that hard-wired factory machine to a programmable robot, able to manufacture anything using common process building blocks. Once a warrior is familiar with our building blocks, he won't need to choose between power and flexibility. Examples of Mission Behavior Specification include:

- High-level requirements for the decision support agent, as obtained from a subject matter expert familiar with unmanned ground vehicle missions.
- Cooperative reconnaissance / security
- Network adaptation for assured coms.
- Adjustability to component failures as a team (fault tolerance)
- Team protection
- Multiple optimum observation points
- Surveillance of moving urban targets

A range of mission planning and specification tools are available for the user to generate the mission, using available terrain data. The decision support agent will assist in the selection of the most appropriate interface by understanding the user's profile, the nature of the mission, and other relevant factors. In this section we describe several ways in which missions can be generated within ITATSS.

One such component resides within the usability tested MissionLab system. A pressing problem for robotics in general is how to provide an easy-to-use method for programming teams of robots, making these systems more accessible to the average user. Toward that end, the MissionLab mission specification system has been developed. This visual programming interface (cfgedit, a component of MissionLab) allows either a human operator to configure one or more robots to accomplish a mission specific task using a usability tested and validated interface, or instead allows the decision support agent to formulate the mission plan on its own under higher-level human guidance. Using Georgia Tech's MissionLab system as a point of departure, a wide range of tactical behaviors, skills, and scenarios have already been incorporated for testing in designated urban scenarios. This was easily accomplished due to the modular, vehicle-independent, and architecture-independent nature of this usability tested pre-mission software system. MissionLab is currently in its sixth major release having started development in 1994. Funded by and used within DARPA's Real time Planning and Control, Demo II, Tactical Mobile Robotics, Mobile Autonomous Robotic Software, MARS 2020, UGCV, and FCS-C programs, it is already geared for military applications. The versatility of this system makes it readily taskable for a wide range of missions, from small urban robots, to scout missions, to explosive ordnance disposal tasks, and many others. MissionLab serves as a rapid prototyping mechanism as well for new tactical behaviors that will be developed during this effort.

MissionLab provides a visual programming environment where operators do not need to write any code whatsoever, but rather by using menus, icons, and connections compose complex military missions in a natural way. Its ability to support component reuse at the behavior, mission, architecture, and vehicle levels also provides significant time savings in the field where previously created missions simply need to be re-parameterized as needed. MissionLab also contain reinforcement learning and case-based reasoning mechanisms developed under the DARPA MARS program, providing inherent performance improvement capabilities through machine learning principles at run-time if desired.

MissionLab has been designed from its inception as a means for specifying operations for entire teams of robots in an easy to use manner. After the mission has been specified, tasking to individual robots occurs automatically. By permitting contingency planning during the specification process, dynamic changes in team performance and responsibility can occur during mission execution. These tasking capabilities apply equally well for both heterogeneous teams of robots as homogeneous, and can be readily extended not only to disparate UGVs but UAVs as well (our experience in the Army's Autonomous Scout Rotorcraft Testbed Program provides a deep understanding of UAV related issues in the context of military operations and integrated systems.

An agent-oriented philosophy is used as the underlying methodology, permitting the recursive formulation of entire societies of robots. A society is viewed as an agent consisting of a collection of either homogeneous or heterogeneous robots. Each individual robotic agent consists of assemblages of behaviors, coordinated in various ways. Temporal sequencing affords transitions between various behavioral states that are naturally represented as a finite state acceptor. Coordination of parallel behaviors can be accomplished via fusion, action-selection, priority, or other means as necessary. These individual behavioral assemblages consist of groups of primitive perceptual and motor behaviors, which ultimately are grounded in the physical sensors, and actuators of a robot. An important feature of MissionLab is the ability to delay binding to a particular behavioral architecture (e.g., schema-based, MRPL (used in Demo II), ROCI and Player Stage (used in MARS 2020)) until after the desired mission behavior has been specified. Binding to a particular physical robot also occurs after specification, permitting the design to be both architecture- and robot-independent.

MissionLab's architecture appears on the left of Figure 10. Separate software libraries exist for abstract behaviors, and the specific architectures and robots. The user interacts through a design interface tool (the configuration editor) that permits the visualization of a specification as it is created. The right side of Figure 8 illustrates an example MissionLab configuration that embodies the behavioral control system for one of the robots capable of conducting hazardous waste (explosive ordnance) disposal operations that was used in our usability tests. The individual icons correspond to behavior specifications, which can be created as needed or preferably reused from an existing repertoire available in the behavioral library. Multiple levels of abstraction are

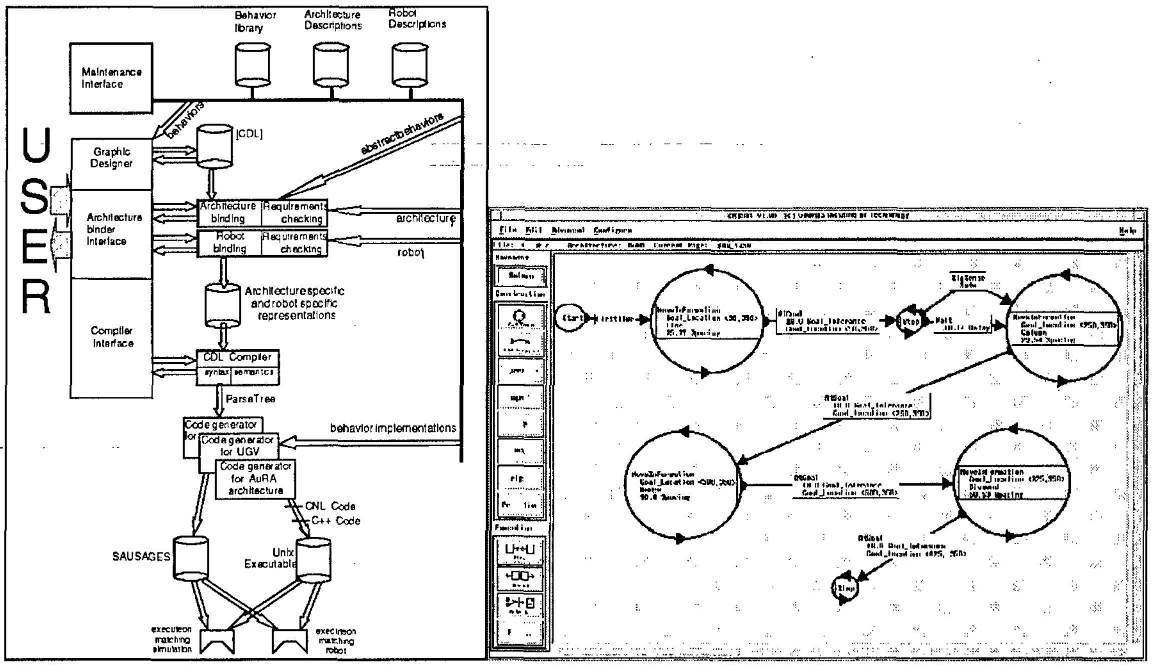


Figure 10. (Left) MissionLab architecture. (Right) Hazardous waste disposal mission specification.

available, which can be targeted to the abilities of the designer, ranging from whole robot teams, down to the configuration description language for a particular behavior, with the higher levels being those easiest to use by the average Soldier.

After the behavioral configuration is specified, the architecture and robot types are selected and compilation occurs, generating the robot executables. These can be run within the simulation environment provided by MissionLab or, through a software switch, they can be downloaded to the actual robots for execution. MissionLab is freely available via the world-wide web at: http://www.cc.gatech.edu/aimosaic/robot-lab/research/MissionLab.html. Figure 11 depicts another mission example along with MissionLab's run-time console.

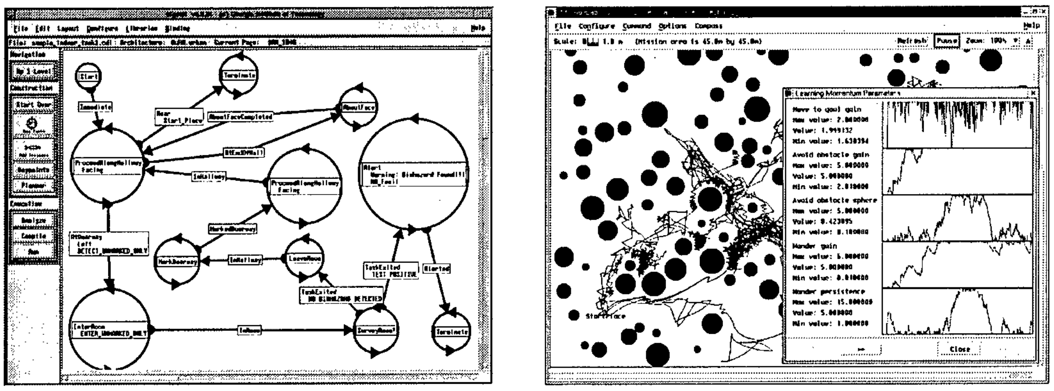


Figure 11: Georgia Tech's MissionLab will enable mission-specific planning (left) and allow us to use simulation tools (right) for validation and verification.

Although MissionLab has since its creation used a human operator as the means for mission specification, the ITATSS decision support agent can also provide additional support for the generation of the overall mission plan, further reducing the cognitive requirements on an operator in the field. Case-based reasoning tools have already been applied to that end at Georgia Tech (Endo et al., 2004) as part of the DARPA MARS program, and thus we expect little difficulty in interfacing the high-level decision support agent to the configuration description language mission plan definition, discussed in section 5.5.

Another interface available to the user for mission planning and specification was developed by SAIC as part of the Demo III program. It is oriented towards missions involving waypoint navigation and maneuver operations for UGV-based scout operations. As an example, the "Add Measures" and "Add Units" dialog panels, shown in Figure 12, allow the operator to annotate the map display with a wide variety of military symbology designating Tactical Graphics for Command, Control and Maneuver, and Warfighting Symbols for Units and Equipment. The Graphical Situation Display (GSD) software package is used to provide standard icon identifiers as detailed in the DOD Interface Standard Common Warfighting Symbology (MIL STD-2525A). The Unit graphics can be designated as friendly, enemy, neutral, or unknown, and the Echelon can be selected. In addition, the operator can identify the graphical symbology with labels. These panels are primarily used in the pre-mission planning phase to provide the most up-to-date information about the tactical situation and to place control/decision points on the map.

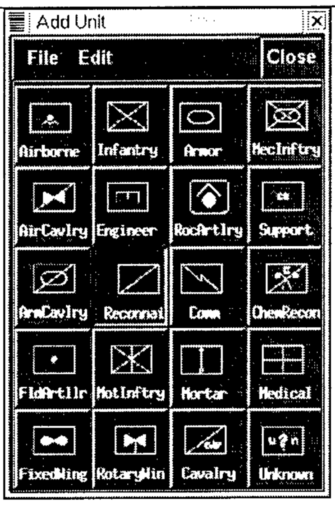
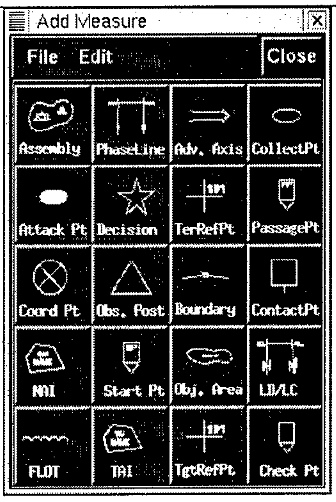
 Add Unit File Edit Close Airborne Infantry Armor MecInftry AirCavlry Engineer RocArtlry Support ArmCavlry Reconnai Comm ChemRecon FldArtllr MotInftry Mortar Medical FixedWing RotaryWin Cavalry Unknown	 Add Measure File Edit Close Assembly PhaseLine Adv. Axis CollectPt Attack Pt Decision TerRefPt PassagePt Coord Pt Obs. Post Boundary ContactPt NAI Start Pt Obj. Area LD/LC FLOT TAI TgtRefPt Check Pt
a) Add Unit Dialog Panel	b) Add Measure Dialog Panel

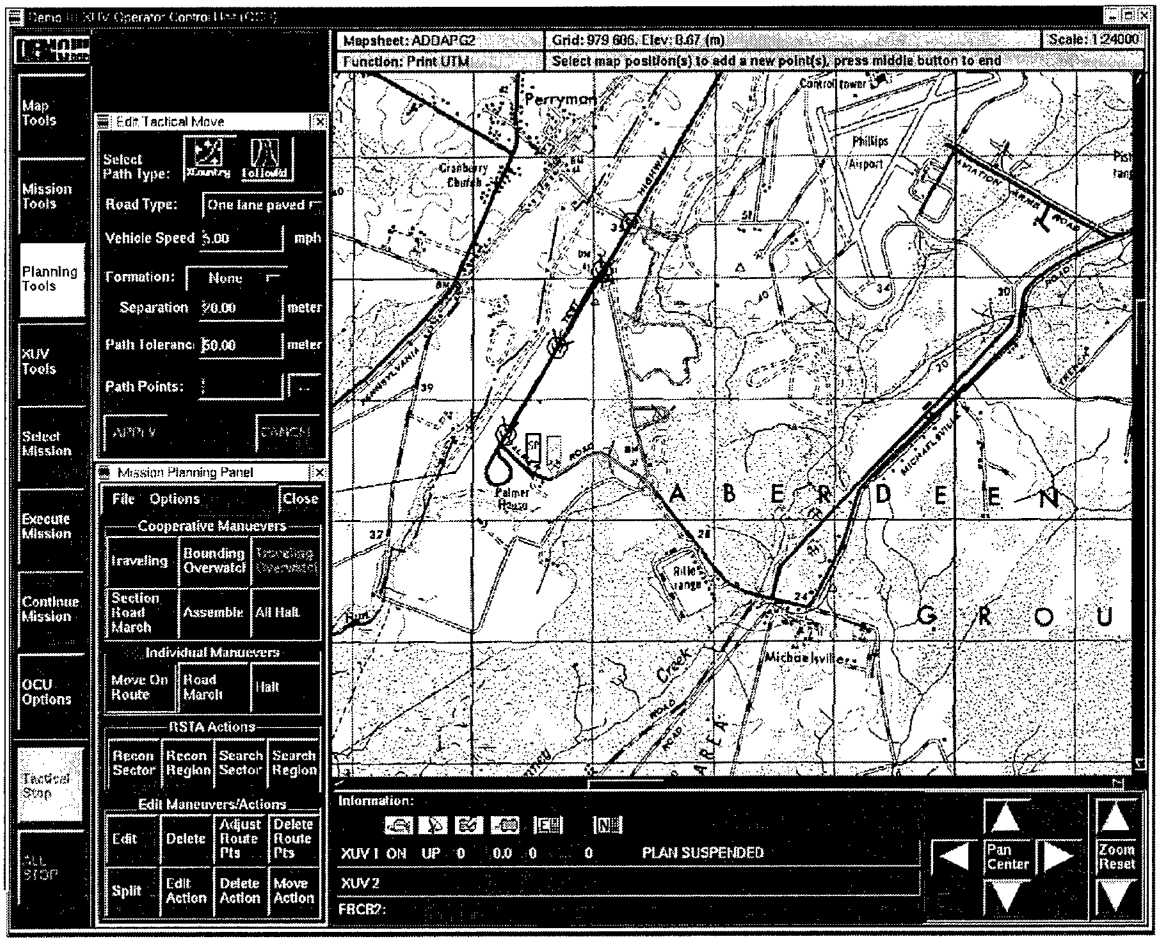
Figure 12. Mission specification tools.

In addition to the manual controls, the SAIC-developed OCU provides a message-based interface through which external systems can affect the Command and Control and Situational content of the OCU display. It is envisioned that in future demonstrations, this capability will form the basis for the Army Battle Command interfaces. This same interface will support the development of real-time tactical behaviors that react to changes in the battlefield situation.

Mission Planning is the process of instructing the UGVs to perform tasks and operations in support of the overall mission objectives. Planning controls are provided which allow the operator to construct UGV plans by graphically drawing on the map display. The operator selects from a pallet of allowable maneuver tasks and RSTA actions he would like the UGVs to perform and specify the details of these actions graphically. During the graphical entry process, text based popup menus are activated allowing the operator to adjust maneuver parameters such as speed and route tolerances.

For Demo Alpha (part of the Demo III program), the basic controls allowed the operator to designate maneuver operations first and then assign RSTA actions to the mobility behaviors. Maneuver operations could be specified for individual vehicles independently or for a team of vehicles operating cooperatively. For a single vehicle the allowable set of maneuver operations is; to perform a tactical move along a specified route, to perform tactical or operational halts, or to conduct tactical roadmarch operations with the C2 (command and control) vehicle. In the case of tactical roadmarch, the specified UGV would travel in formation with the OCU. Cooperative maneuver tasks included the ability to travel in formation as a team or to perform bounding overwatch or traveling overwatch maneuvers. Additional cooperative operations include the ability to task the UGVs to assemble at a common map location, to stop as a team, or to conduct tactical roadmarch operations as a team with the C2 vehicle. In the case of the Traveling maneuver, the mission planner would automatically generate the sequence of actions leading up to the traveling maneuver. Bounding Overwatch and Traveling overwatch were not implemented as cooperative maneuvers for Demo Alpha, although these maneuvers could be specified by constructing sequences of individual UGV functions. Color and position were used to distinguish between plans for different vehicles. The color scheme was consistent with the one used to distinguish between different UGVs.

Allowable RSTA (Reconaissance, Surveillance, and Target Acquisition) operations include the ability to perform automated searches or to gather imagery over a specified azimuth sector or map region. In the case of automated search, the UGVs employed Automated Target Detection and Recognition (ATD/ATR) technology and acoustic sensing technology to detect targets and enemy activity. Automated search can be performed either while on the move or from stationary RSTA overwatch positions. In the later case, the UGVs employ an advanced image processing technology to automatically construct large mosaic images while the RSTA pan/tilt mechanism slewed the sensor over the specified region. Figure 13 shows a representative mission plan and the accompanying mission planning control panels.



Mapsheet: ADDAPG2 | Grid: 979 686, Elev: 8.67 (m) | Scale: 1:24000

Function: Print UTM | Select map position(s) to add a new point(s), press middle button to end

Road Type: One lane paved

Vehicle Speed 5.00 mph

Formation: None

Separation 20.00 meter

Path Tolerance 50.00 meter

Path Points:

XUV1 ON UP 0 0.0 0 0 PLAN SUSPENDED

XUV2

FRCB2:

Figure 13. Mission planning tools.

In its base configuration, the OCU provided the means to manually specify all actions for the UGVs. Optionally, the operator could activate automated planning tools to assist in the planning process. These tools include a route planner that is sensitive to traversability constraints, communication coverage constraints, and enemy threat masks as input from the map analysis tools and a tactical behavior planner that assists in the automation of specific tactical skills and tasks. For Demo Alpha, the tactical behavior planner would automatically insert stop and report points at phase line crossing and would automatically compute the sequence of steps to be performed when Establishing an Observation Post. Specifically, the planner would ensure that the area containing an Observation Post was first scanned from a concealed location before proceeding to the overwatch location near the OP. It is envisioned that the automated planning capabilities be extended to include addition tactical skills as the Demo III program progresses. However, the overall goals of the Demo III program are to implement as many of the tactical tasks and skills in the run-time elements of the software. As the run-time software become more capable the need for planning within the OCU is likely to reduce.

Mission Planning can be performed off-line, prior to mission execution or while the UGVs are actively being controlled in the mission environment. Typically, the mission planning tools remain visible to the operator anytime the UGVs are moving. This allows the operator to rapidly re-task the UGVs or perform plan adjustments as needed. In the off-line case, the OCU provides the capability to store and retrieve mission plans for future execution. In both cases, the decision support agent will combine knowledge of the current situation with knowledge of the user's personality profile to recommend the appropriate user interfaces for different types of mission planning.

4.2.1 Augmented mission planning

In ITATSS, these mission-planning systems will be supplemented by the DSA, which must contain knowledge specific to mission planning issues. In a typical scenario, after receiving a mission, a human operator would plan the robot routes and maneuvers and then rehearse them. Planning would take into account any potential encounters during the mission, such as interactions with the enemy or other units. Planning would also take into account constraints on the robot's mission payload (weapons and sensors), such as identifying potential "dead spots" for the robot's weapons and communications devices, and managing the abilities of the robot's sensors to see vertically and horizontally. Planning would also involve planning travel with the robot and finding potential dismount points for the robot. Next, the operator would plan particular activities to take place at regular timed intervals or at event points along the robot's route. The DSA must incorporate knowledge for each of these facets of mission planning.

In order to develop the plan there may be a variety of different possible user interfaces or tools available to the user. Selection of the best user interfaces would depend on mission and situation information, as well as the preferences of the user as defined in the user profile. The DSA will incorporate knowledge to assist the user in making the best choice for mission planning tools and interfaces.

Once the plan is complete, it is important that the operator be able to remember the plan and execute it as efficiently and flexibly as possible. Ideally, given adequate time and other resources, this would involve some amount of mission rehearsal. However, with the addition of a decision support system, we also introduce the possibility of supplementing mission rehearsal with on-line memory aids as the mission executes. Thus, the DSA must also include knowledge of how to annotate the plan with "triggers" that will invoke additional decision support during mission execution. According to the initial DSA requirements analysis we have performed with our subject matter expert, the DSA and associated user interfaces should provide:

- Support for mission planning, including identification of potential routes, three- and four-dimensional visualization tools to display terrain maps annotated with relevant tactical information, displays providing graphical representations of assumptions and beliefs about the mission environment, and on-demand explanatory facilities to help the operator identify, distinguish, and select appropriate route and event points.
- Support for mission rehearsal, providing a realistic environment for simulating the execution of the mission in a manner as close as possible to the operational environment that will be used for mission execution. Providing realistic and effective mission rehearsal support suggests the type of modular design we are proposing in ITATSS, allowing the same system to interface with real and simulated robots and environments.

4.3 Run-time situational awareness and control

Although ITATSS robots can operate at either at autonomous, semi-autonomous, or teleoperated levels, considerable care must be taken in providing suitable interfaces to both the human operator and the automated decision support agent. We intend to provide these interface capabilities at multiple levels, drawing heavily upon our previous work for other related DARPA programs, leveraging their results. This section enumerates and illustrates these capabilities provided to the ITATSS architecture.

Apart from providing a basic required teleoperation capability, we have several fundamental goals:

- Developing a reliable and effective human-centric user interface,
- Augmenting teleoperation with semi-autonomous behaviors that strengthen the probability of mission success
- Incorporating elements of *team teleautonomy*, enabling robotic units to function as a cohesive unit.

Our research approach has taken the somewhat unusual stance of starting with a fully autonomous capability (embodied in our MissionLab mission specification system) and then engineering more effective run-time teleoperation capabilities. In a sense, we are reducing the on-board intelligence of the robots to incorporate and complement human intelligence.

Varying levels of autonomous capabilities are possible for mobile robots, ranging from simple teleoperation up to complete "human-like" intelligence. Even within the realm of teleoperation, there is a spectrum of capabilities (see Figure 14). Teleoperation, by putting a "human in the loop," provides enormous flexibility in a robotic system and will be a key aspect of inserting robots into military applications while autonomy continues to evolve. But direct teleoperation is really just as dependent on the human operator and the communication channel as is ordinary R/C control – it assumes that the operator has a reliable connection (and high bandwidth) in order for effective operation to continue. We have already explored methods for reducing this dependence on the operator/commlink components, replacing them with increased reliance on the robot itself, including:

- Safeguarded teleoperation – preventing the operator from making mistakes through the use of robotic sensors and appropriate protective behaviors.
- Designated waypoint teleoperation – allowing the robot to continue making progress toward one or more designated waypoints in the absence of control signals, *especially* when the communication link drops out
- Alert-based teleoperation – permitting semiautonomous operation of the robot with a variety of complex behaviors, alerting the operator to intervene only when certain designated triggers occur.

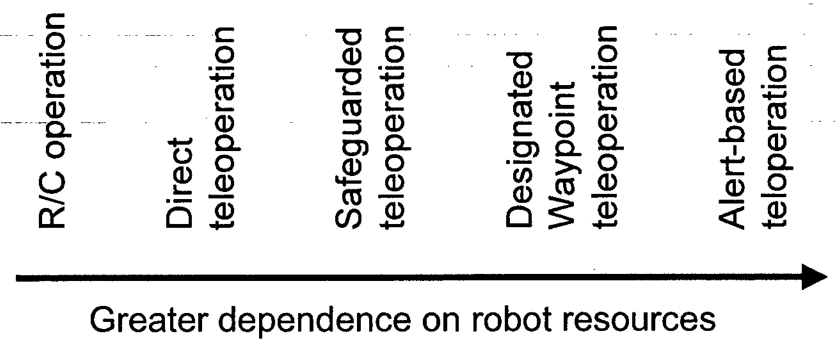

R/C operation
Direct teleoperation
Safeguarded teleoperation
Designated Waypoint teleoperation
Alert-based teleoperation
Greater dependence on robot resources

Figure 14. Spectrum of teleoperation.

Extensive research has been conducted in teleoperated control of robotic platforms for a broad range of perspectives, including system design, human factors, and control architectures. We briefly review a very small sample of such studies that are particularly relevant to this work. Sheridan's (1992) seminal studies in human-controlled robotics have served as the basis for understanding the fundamental limitations of human performance and the design of supervisory control systems. Krotkov's (1996) research in safeguarded teleoperation added intelligent control to supplement the operator's limits. Fong, Thorpe, and Baur's (2001) work has moved teleoperation down to systems capable of operating from Personal Digital Assistants. Other groups (e.g., Halme et al., 2000) have studied camera placement and its impact on operator performance in driving tasks. Various levels of operator interaction within an autonomous system have also been referred to as adjustable autonomy (Musliner, 1999).

One of the goals of the ITATSS human-machine interface is to encourage the evolution of the user's role from controller to advisor to participant to advisee. Current robotic systems usually force the user to be actively in control of the system, which is cognitively taxing and limits their ability to operate the system. One design goal is to make the end-user less of a controller and more of a participant with the autonomous system, perhaps leading towards interaction at a level similar to that found in advanced avionics systems (e.g., Boeing 777), modern flight traffic control systems, automated monitoring systems (e.g., nuclear power plants), or even simply riding a horse that knows it's way home. Ultimately the operator should be queried only where human intelligence is most important: for target engagement, for opportunistic reasoning, for alert-based failure handling and the like. By doing so, this will enable the operator to handle multiple robotic systems more easily while concurrently reducing cognitive load, enabling longer term interaction with the system with reduced fatigue. As human attention is inherently serial in nature, it is crucially important to direct attention in as simple and non-taxing manner as possible. By further engaging ITATSS's decision support agent to assist in high-level decision making, the ability of the operator to make the right decision at the right time will be increased.

The net result of our approach will be an evolution of display technologies and underlying software presentation techniques that will continuously, over time, throw more and more responsibility onto the autonomous system and less and less on the human operator. Operator training methods and selection principles will be concurrently studied to ensure that the right people with the right skills and the right training are those who are tasked to operate these advanced robotic systems.

We now look in turn at three main interfaces for operator-robot management ITATSS: the teleoperator interface, teleautonomy, and visual mission specification.

The primary interface to the UGV system is through the Operator Control Unit (OCU). Through interactions with the OCU, the OCU operator tasks the UGVs to perform mission related goals issued by the platoon leader. Each OCU operator may control a team or squad of 1 to 4 UGVs.

4.3.1 OCU design

The OCU utilizes standard military map databases, must be sufficiently robust for Soldier operations, and must be capable of planning missions for all terrain types including primary roads, secondary roads, trails and cross-country maneuvers. The design should support multiple RSTA missions, however should be sufficiently modular to be reconfigured for alternative military missions. The OCU design provides for interfaces with Force XXI Battle Command Brigade and Below (FBCB2), an element of the Army Battle Command system, which enables the UGV system to receive the current battlefield situation, receive commands and generate reports digitally. The OCU should be able to integrate with systems used by the modeling and simulation (i.e. OTB). In effect all interactions with the outside world are funneled through the OCUs, as shown in Figure 15.

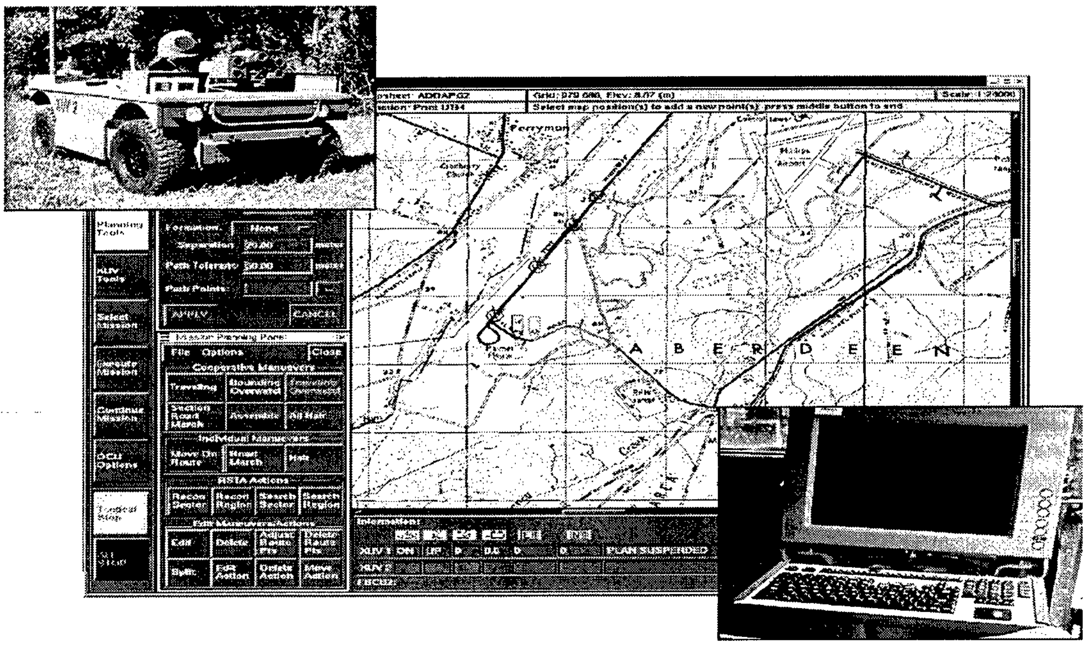
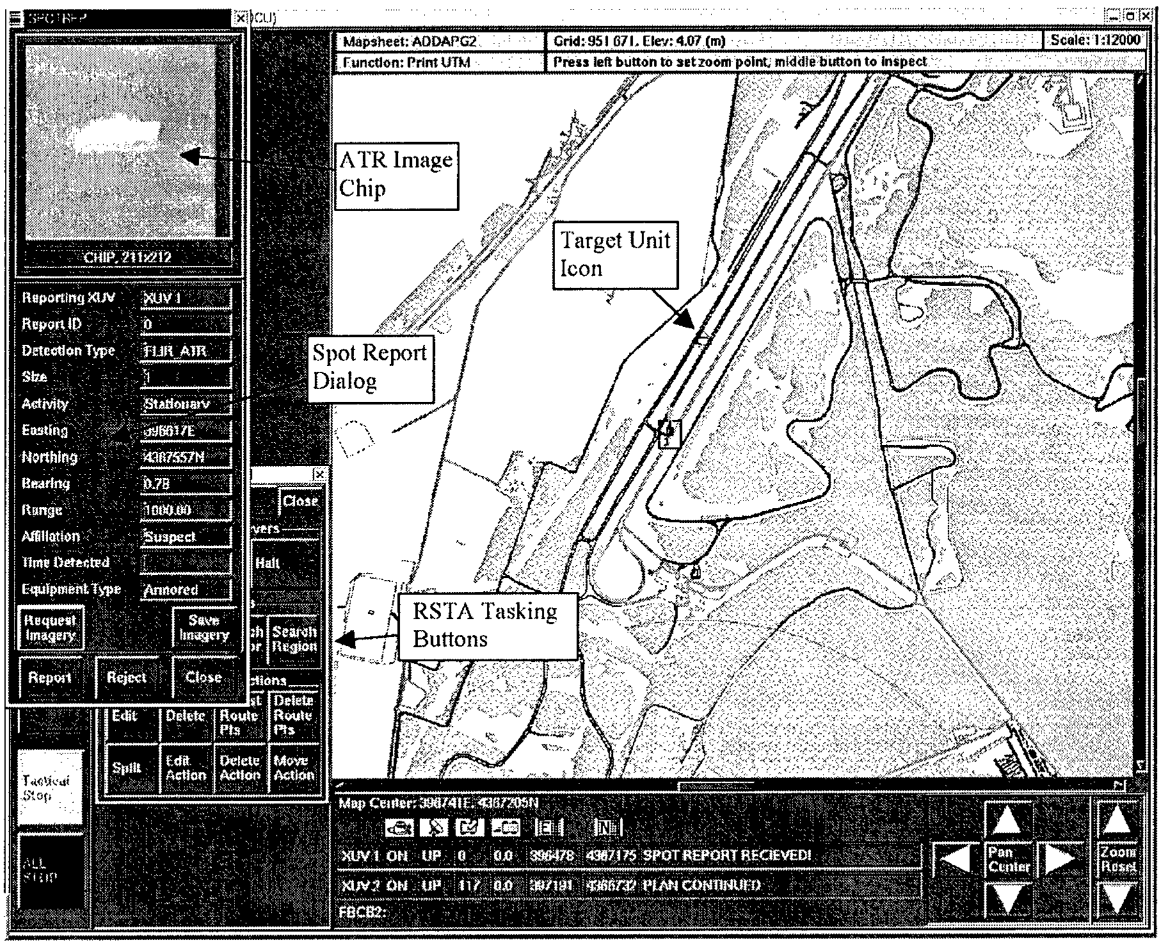


Figure 15. Demo III XUV system and associated Operator Control Units (OCUs).

Following mission planning is mission execution. During mission execution the OCU becomes part of the run-time software architecture. The OCU interacts with the 4D/RCS architecture primarily at what is known as the Section level of the real-time architecture. The section level was initially designed to run on-board the UGVs although for Demo Alpha the section level for each UGV ran onboard the OCU hardware. The primary functions of the section level are to parse the mission plan into sequential steps, and control the execution of these steps in real-time. Other functions support implementations of tactical tasks and skills related to the generation of system status reports at specified time or distance intervals and handling asynchronous events such as RSTA reports and operator requests for information.

During mission execution, OCU controls and displays allow the operator to activate and control individual UGVs or all vehicles simultaneously. The OCU design assigned group controls to the large touch buttons on the side of the panel enabling control of all UGVs while the C2 vehicle was on the move over rough terrain.

Basic controls include the ability to activate a UGV, execute, suspend and resume plan execution. Controls for individual vehicles were available as context sensitive popup menus attached to the UGV icons on the map. During mission execution, UGV status and plan graphics were updated periodically on the OCU map and textual displays. UGV icons showed the current position of each vehicle as well as its speed and direction of travel. The current field of view and gaze direction of the RSTA sensors was also updated dynamically. Additionally the current position of the OCU was continuously updated on the map display. To assist in error recovery, backup controls were also provided from the UGV icon popup menus.



Mapsheet: ADDAPG2
Grid: 951 671, Elev: 4.07 (m)
Scale: 1:12000
Function: Print UTM
Press left button to set zoom point, middle button to inspect

Reporting XUV	XUV 1
Report ID	0
Detection Type	FLIR_ATR
Size	1
Activity	Stationary
Easting	396617E
Northing	4387557N
Bearing	0.78
Range	1000.00
Affiliation	Suspect
Time Detected	
Equipment Type	Armored

```
Map Center: 398741E, 4387205N
XUV 1 ON  UP  0    0.0  396478  4387175 SPOT REPORT RECIEVED!
XUV 2 ON  UP  117  0.0  397181  4386732 PLAN CONTINUED
FBCB2:
```

Figure 16. **OCU RSTA user interfaces.**

The policy of being able to inter-mix mission planning and mission execution proved to be a valuable approach to the Soldier operators at Demo Alpha. One of the performance parameters for Demo Alpha was how quickly the OCU operator could re-task the UGVs when a new mission objective was received from the platoon commander. This occurred several times during the field experiments. In fact following the first day of BLWE trials, the operators almost completely abandoned the concept of planning the entire mission before hand and utilized the tools of the OCU to plan and execute the mission in phases as the battle field situation unfolded.

The information presentation function shall provide the interface between the system feedback and the operator. The information presentation function shall minimize the load on the operator's eyes. This may be achieved through multi-modal presentation and an alerting mechanism. If the operator is not directly commanding the robot, then the visual displays can be inactive. The system shall alert the operator when there is an activity that requires attention. When the operator is alerted the operator's response may indicate the presentation preference. The visual display may provide a hierarchy for providing contextual reference and allow the operator to interact with the system using only the information necessary for the particular activity. The operator shall be able to query the system for information at any time.

The HRI is responsible for managing several forms of information. The method of representation for each set of data depends upon the nature of the information and its access requirements. HRI information shall include but is not limited to operator configuration data, employment plans, rule bases, and operational information.

Operator configuration shall include the information necessary to maintain HRI configurations for a set of operators to control multi robots. The configuration database shall include a list of operators and the HRI device configuration for each operator. A default configuration shall also be maintained in the configuration database in case an operator has not provided a device configuration. Each operator shall have the ability to make modifications to his/her own device configuration.

A Plan Database may be used to store plan component representations. The plan database may contain the implementation details of all of the behaviors, roles, and plan elements known to the system. When creating a specific plan for tasking a robot or team of robots, the operator may draw from this database or add created elements to the database. This database is meant to contain generic plan elements only, not the binding environmental factors of a specific instantiation of a plan. This information is held in a separate structure and used as the basis for interpreting information feedback from the robot team.

A set of rule bases may be used to support the information processing function. These rule bases define the constraint models for the human robot interaction process. Throughout the use of the HRI system, annotations may be made here regarding operational knowledge gained from using the HRI system. Knowledge represented here shall include the following:

- rules defining failure conditions for platform mobility
- rules defining how to fuse disparate pieces of robot feedback information
- operator cognitive models defining information feedback and C2 constraints
- operator interaction error models

Information contained in these knowledge bases is intended to aid in the analysis of the human robot interaction process. This information may be updated automatically by the HRI as a result of failure analysis processes, or may be updated by the operator to contain specific elements of operational knowledge obtained through the use of the system.

The HRI shall maintain operational information and allow an operator to review plan progress. In this review process, the HRI shall allow the operator to interact with the C2 structure and receive operational and status feedback at the plan, role, behavior, and task levels. Alerts, as the highest level of operational information, shall be presented to the operator as received from a robot or the HRI.

An information management plan shall be implemented to govern the storage of information as well as the purging of older information from the system. The HRI shall take advantage of higher level data abstractions provided by the C2 structure to minimize the raw sensor information that must be stored.

The Information Management subsystem may be capable of providing the operational context of a particular robot after it has failed in some way. An information management tool may allow the operator to “replay” a platform’s activities, including the necessary sensory information received over a certain period of time to provide the contextual awareness needed for an operator to understand the state of a particular robot or team of robots.

4.3.2 Teleoperator interface

Developed under DARPA’s FCS-C program, this interface provides both a windshield and dashboard capability for controlling a robot (see Figure 17 and Figure 18). A subset of these interface capabilities was recently used as part of a successful demonstration of the DARPA MARS 2020 Program at Ft. Benning in December 2004.

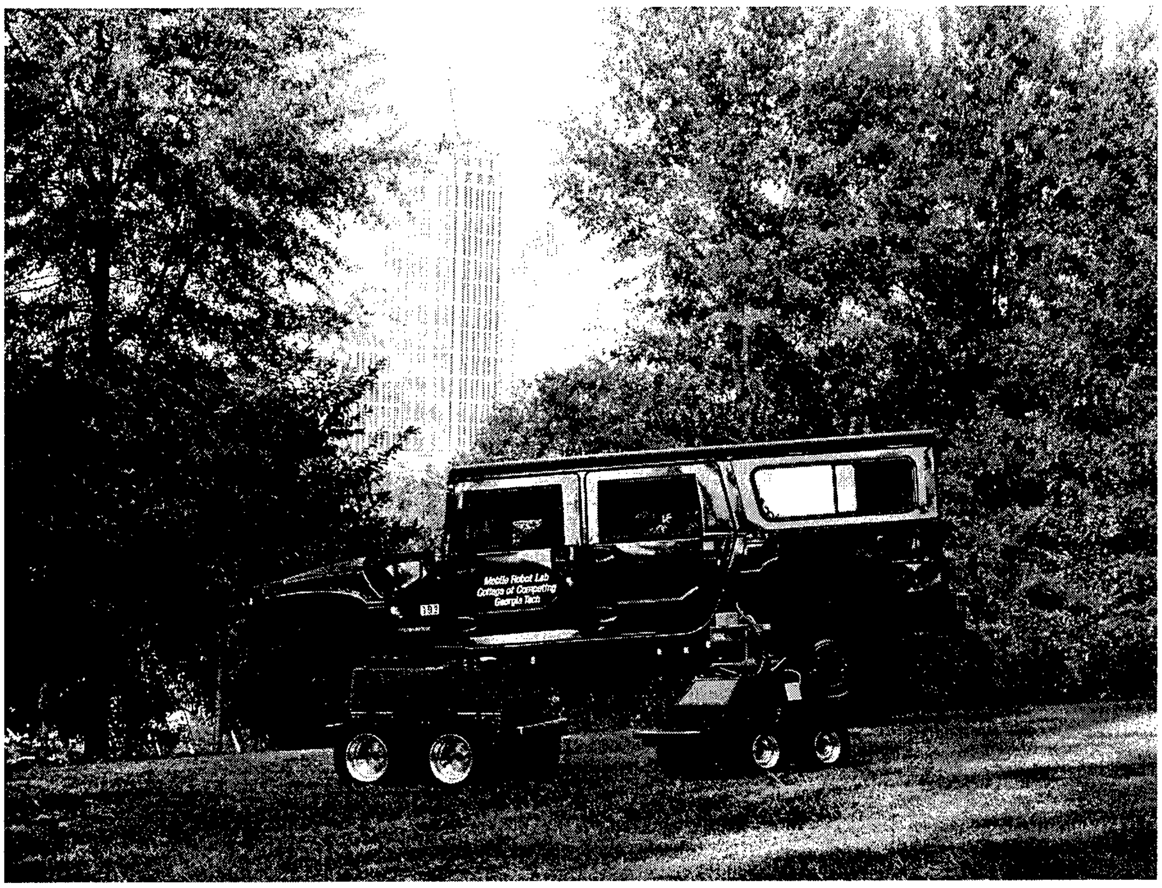


Figure 17. Base platforms used for robot team for FCS-C interface testing and development.

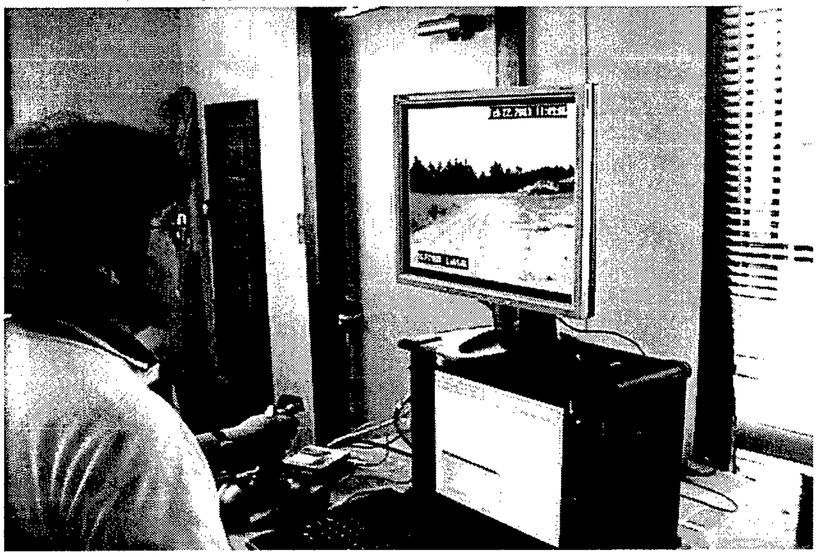


Figure 18. Prototype teleoperation workstation.

The remainder of this section describes the architecture and interface details of the user interface that allows seamless teleautonomous operation of a single robot. Many of these components exist as off-the-shelf technology developed at Georgia Tech under different DARPA programs. The complete teleoperation system consists of two parts: the dashboard and the windshield. Each is meant to be displayed on a separate video screen analogous to a driving scenario. The windshield contains video data coming from the robot, along with other vital information such as speed, orientation, and obstacle detection warnings. The dashboard, on the other hand, contains the MissionLab console, along with other vital information such as battery and communication status. In addition, dynamic waypoint and communication loss trigger features are available, which allows the user to add and skip waypoints in real time.

4.3.2.1 Teleoperator dashboard

The dashboard has three primary features that enable easy teleautonomous operation. One is the GUI, which shows useful information to the user and allows the user to navigate and control the robot in real time. The second is the dynamic waypoints feature, which allows the user to add or skip waypoints on the fly. Finally, in the event that communication is lost between the robot and the host, an appropriate trigger was added, whereby, if so specified during mission creation, the robot can return to a previous or alternate state or location when communication is lost.

The Dashboard GUI was added onto MissionLab and consists of many parts (see Figure 19). The main central window is the MissionLab console, which displays the robot's location in map (terrain) coordinates, mission waypoints, and sensor data (sonar, lasers, etc.) In the upper right corner, there is an egocentric view that displays sensor data with respect to the robot. Under it is the Objective data, which displays the coordinates of the current goal the robot is pursuing, as well as Euclidean and angular distance from the robot to that goal. The Target field contains the coordinates of the robot. If the robot supports GPS, it will display latitude and longitude. It also has the current time displayed.

The Battery display shows the current battery voltage, so that the user knows when the robot should start heading back. Beneath it is the Communication display, which shows ping loss (percentage of ping packets lost) and ping time delay (ms). It shows both current numbers, as well as a graph that shows the history of communication quality. This display is useful when there is poor or no communication between the robot and the host computer.

The Deadman Switch that is under the Comm display is used only when the physical joystick is utilized. In order to teleoperate the robot, the trigger on the joystick must always be held down; otherwise moving the joystick does not affect the robot. The E-Stop button is an emergency stop that pauses execution of the robot.

Below the console window lie the teleoperation windows. This consists of several choices that the user can make, such as using the mouse or joystick for teleoperation, as well as a display that shows the current vector applied to the robot by the user.

Finally, there are a few buttons that are on the lower left. The Skip Waypoint button skips the current waypoint that the robot is moving towards, whether it was dynamically created or pre-specified in the mission. The Disable Telop button disables all teleoperation and reverts to autonomy. The Send Bad Comm button tricks the robot into thinking that there is bad communication between it and the host, which was useful for debugging and testing purposes, but of little value for real operation.

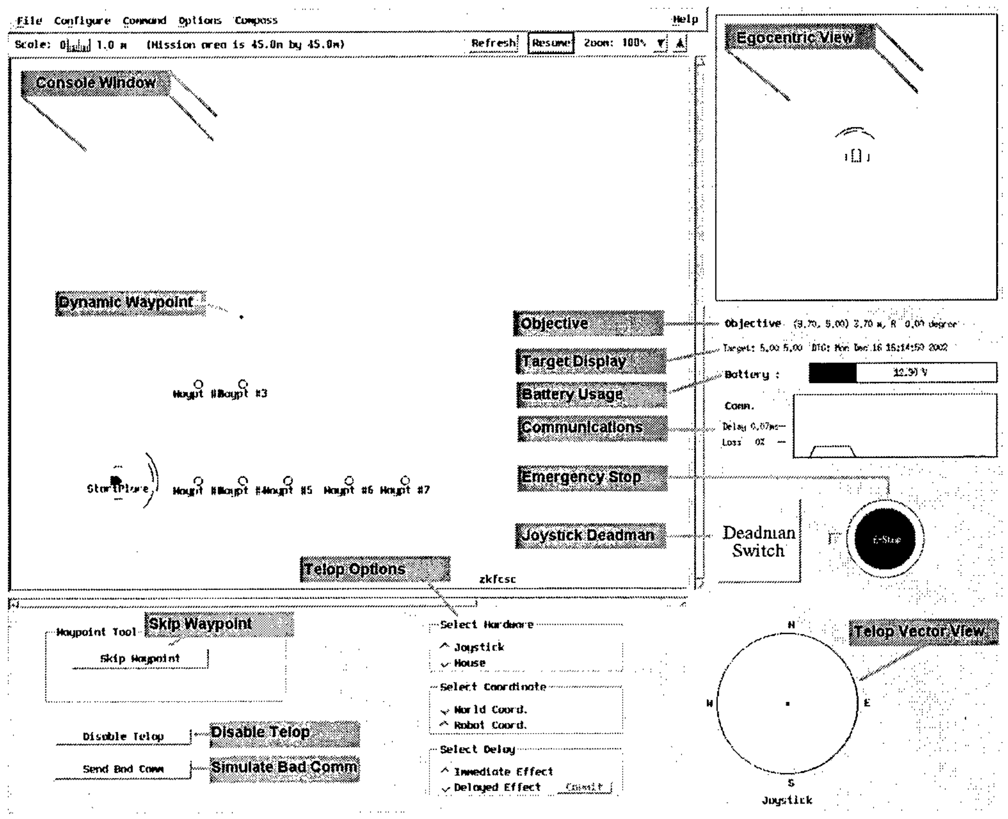


Figure 19. Teleoperation dashboard GUI.

4.3.2.1.1 Dynamic waypoints

Although the user specifies waypoints for the robot to follow when creating the mission, many situations can arise where those waypoints would need to be changed or removed based on incoming intelligence. It is also possible that the high-level ITATSS decision support agent could suggest these waypoints automatically to the user for verification. As an example, if an unforeseen obstacle completely blocks waypoint attainment, it would be useful for the user to remove that waypoint so that the robot can continue its mission. For these reasons, the dynamic waypoints feature was added. A user can add a dynamic waypoint by simply clicking on the MissionLab console on the dashboard. If multiple waypoints are added, they are queued and the robot will go to them one by one (FILO: first in last out). The user can also order the robot to skip the current waypoint it is heading towards, including dynamically created ones.

4.3.2.1.2 Communication loss trigger

If communications between the robot and the host is poor, it would be useful if the robot can behave intelligently and take this into account. One way to do this is to add a trigger to the FSA that reacts to bad communication. When creating the mission, the operator can add a transition to various alternate states/tasks if communications loss occurs. For example, the robot can autonomously retrotraverse (return to a previous location), assuming that communications will be better there. Once communications is regained, the user can use dynamic waypoints and waypoint skipping to redirect the robot around the area known to have bad communication characteristics. Depending on the situation, different strategies might be more useful than others. Considerable work in communication-sensitive behavior and planning undertaken as part of the DARPA MARS 2020 program has potential relevance here.

4.3.2.2 Teleoperator windshield

The windshield is somewhat more straightforward to describe as it provides an enhanced real-time video display to the teleoperator (see Figure 20).

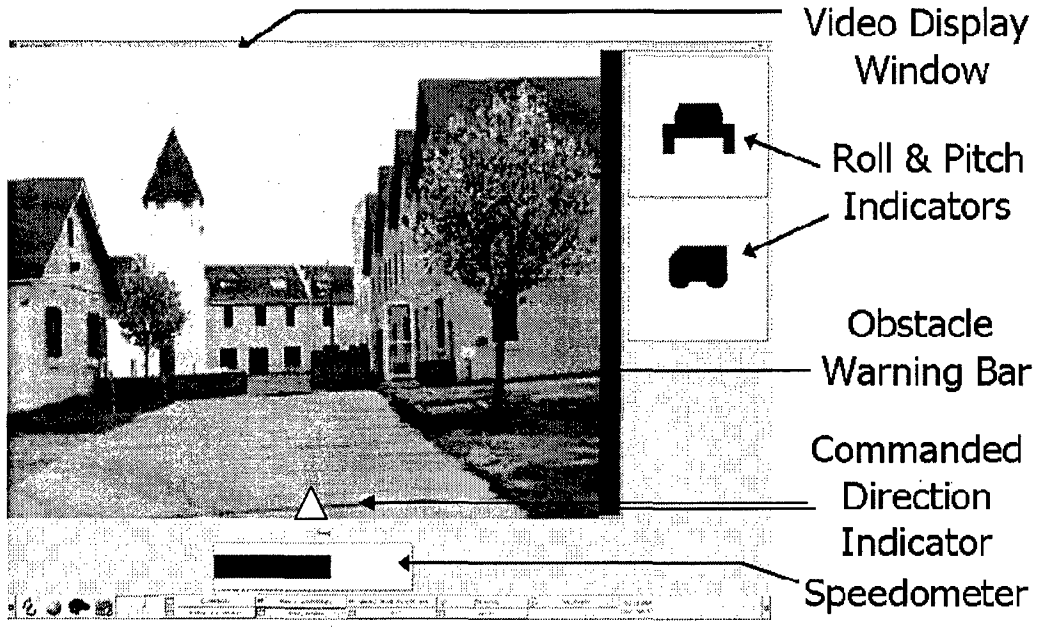


Figure 20. Windshield screenshot.

The windshield's functionality and features include:

- Display video from the robot
- Make the operator continuously aware of the robot's status and its environment.
  - Show feedback of commands to robot
  - Warn operator of obstacles near the robot but out of view of the camera
  - Show Roll and Pitch of the Robot
  - Show speed of robot
- Commanded Direction indicator
  - Red triangle moves right and left according to input from the joystick
- Obstacle Warning Bars
  - Red rectangular strips appear on right or left of video screen when obstacles are located to the left or right of the robot, out of the field of view of the camera.
- Roll & Pitch Display
- Speedometer

4.3.2.3 Teleautonomy interface

Developed as part of DARPA's UGV Demo II program, this system allows either the human operator or the ITATSS decision support agent to directly affect the behavioral control of one or more of the available robotic agents.

Also highly relevant is the notion of team teleautonomy, developed at Georgia Tech, where a single operator can easily control an entire team of robots through the real-time introduction of an operator's or agent's intentions to the ongoing operation of an autonomous robotic team. We have already developed the software that provides this capability to the end-user. It functions in two distinctly different ways:

1. *The operator as a behavior.* In this approach a separate behavior is created that permits the commander to introduce a heading for the robot or robot team using an on-screen joystick (Figure 21) (this can be easily replaced by a voice command system or other no-hands control method). This biases the ongoing autonomous control for all of the robots in a particular direction. All other behaviors remain active, including, for example, obstacle avoidance, goal seeking, and formation maintenance. The output of this behavior is a vector, which represents the operator's directional intentions and strength of command. All of the robotic team members have the same behavioral response to the operator's goals and the team acts in concert without any knowledge of each other's behavioral state.

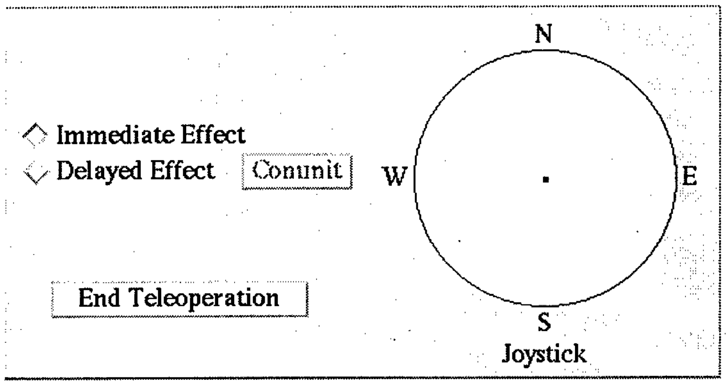


Figure 21. On-screen joystick.

2. *The operator as a supervisor:* With this method, the operator is permitted to conduct behavioral parametric modifications on the fly. This can occur at two levels. For the knowledgeable operator, the low-level gains and parameters of the active behavioral set can be adjusted directly if desired, varying the relative strengths and behavioral composition as the mission progresses. For the normal operator, behavioral traits ("personality characteristics") are abstracted and presented to the operator for adjustment (Figure 22). These include such things as aggressiveness (inversely adjusting the relative strength of goal attraction and obstacle avoidance) and wanderlust (inversely varying the strength of noise relative to goal attraction and/or formation maintenance). These abstract qualities are more natural for the operator unskilled in behavioral programming and permit the concurrent behavioral modification of all of the robots in a team according to the operator's wishes in light of incoming information.

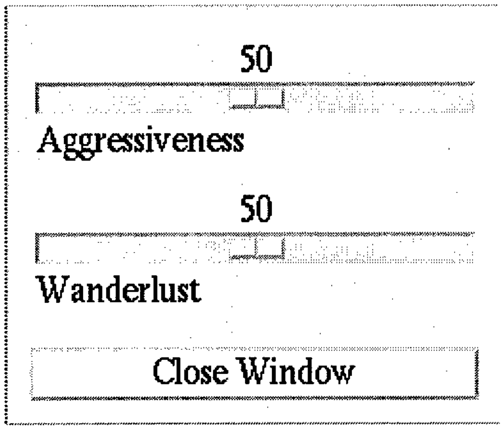


Figure 22. Personality teleautonomy window.

Both directional and personality team teleautonomy control have been fully integrated into the MissionLab system, and tested on teams of Denning, Nomad, Pioneer, and ATRV robots as well as in simulation, and are available for integration into ITATSS.

4.3.3 Human interaction with decision support agent

The decision support agent's primary mode of assistance will be during mission execution, helping to augment the run-time situational awareness of the operator, and suggesting potential courses of action and tradeoffs in response to significant events in the mission plan (as well as significant events not anticipated by the mission plan).

Our subject matter expert assisted us in creating an initial task analysis to define the performance knowledge relevant to mission execution. According to this analysis, the ideal during execution is to follow the mission plan as closely as possible. Realistically, however, the mission involves contingencies, and the operator must be prepared for a battlefield that changes in expected and unexpected ways. Thus, analysis of the task reveals that there can also be significant decision making during the course of the mission, even during a preplanned mission. Depending on the mission and the situation there will be different types of decision points. A minimum set of decision points might include:

- When and where to dismount the robot, depending on local security of friendly elements and current status of known enemy locations along the planned route.
- Decision points caused by absence or presence of enemy actions, including possibly continuing movement, calling for fire, changing the route, seeking defensive position, and/or observing enemy activity.
- Whether to establish or continue surveillance, depending on the security of the desired observation point.
- Decision points caused by acquisition of new situational information, such as new enemy locations or activities, which may lead to continued observation or possible maneuvering to improved observation points (with a better view and/or better cover and defensive positioning).

Our initial high-level task analysis identifies many of the types of activities that the robot operator must engage in during mission execution (as well as planning and rehearsal). It also provides a guideline for which types of decision support might be most useful to the operator. We should stress again that it does not make sense for engineers to try to guess what types of decision support would (or should) be most useful for a robot teleoperator. Building the most useful system depends on designing and testing the support system with actual users. Our subject matter expert has taken a first cut at identifying the types of decision support he would find most useful. Further work on the DSA will implement the first level of the support system, test the initial design using an agent prototype and demonstration scenarios, and further iterate the requirements for the most beneficial types of decision support.

Following the task analysis performed so far, the DSA and associated user interfaces should initially provide the following types of support for mission execution:

- Switchable three- or four-dimensional dynamic view of mission execution, including reminders of planned action at important decision points (such as robot dismount, objective actions, interactions with friendly forces), in response to preplanned contingencies for event responses (such as interactions with enemy or noncombatant forces), or periodic scheduled actions.
- Integrated alert and suggested response system for reaction to unplanned events (such as surprise encounters with enemies or non-combatants, serendipitous achievement of mission goals, or unexpected defeat of mission goals).

- Advisory information on selecting appropriate tools and interfaces to use during mission execution. The DSA should contextualize this advice to changes in the situation and elements of the operators user profile preferences.

5 Functional components of ITATSS

As we have outlined previously, and as Figure 23 reiterates, the ITATSS architecture integrates the goals, awareness, and behavior of three entities: the human operator, the decision support agent, and the robot vehicle platform (or a simulation, for some applications). Each of these entities shares information with the others through the shared knowledge repository. Thus, an important focus of ITATSS' design and implementation will involve the various functional components that provide automated behavior, and that provide communication between the entities and the SKR. In Section 4 we detailed the various user interfaces that the operator will use to interact with the other entities (and the SKR). This section details the other functional components of ITATSS that provide the interfaces between the SKR and the other digital systems.

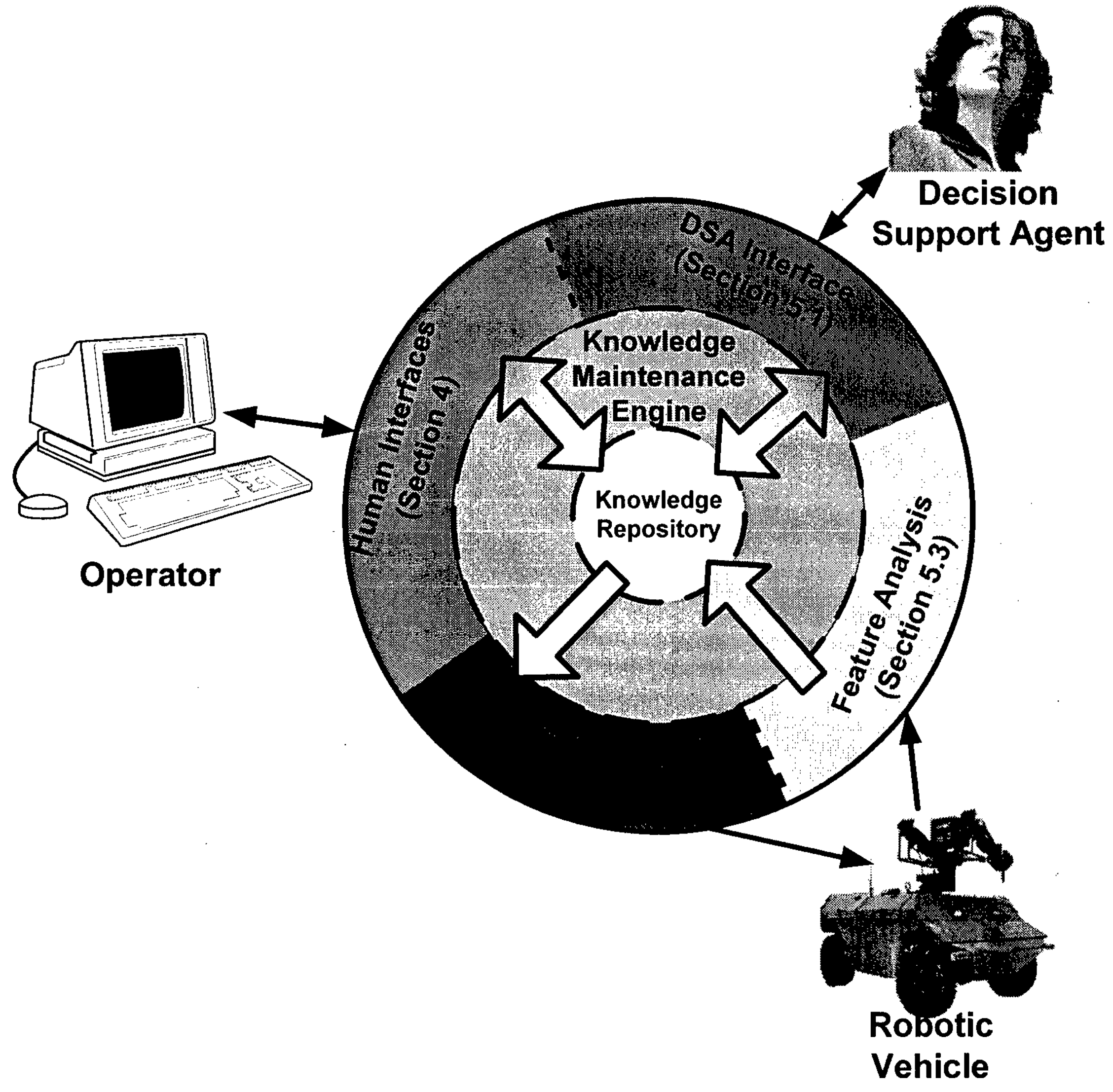

Decision Support Agent
DSA Interface (Section 5.1)
Knowledge Maintenance Engine
Knowledge Repository
Human Interfaces (Section 4)
Operator
Feature Analysis (Section 5.3)
Robotic Vehicle

Figure 23. Functional high-level view of ITATSS entities, functional components, and interfaces.

5.1 Decision support agent

Perhaps the most complex functional component of ITATSS will be the knowledge-intensive intelligent agent system that implements the decision support agent. The decision support agent must incorporate genuine understanding of the operator task, together with knowledge of how to communicate and interact with the operator and robot, as well as pedagogical knowledge for training purposes. In addition, the decision support agent must interact in real time with the robots' sensor systems, digital intelligence-gathering and communications systems, and the human operator, all the while constructing a sophisticated internal model of situational awareness (in the SKR), and displaying key components of this awareness to the operator (via the user interfaces).

Supporting these requirements with long-term cost effectiveness requires a departure from standard software systems development. Theoretically, it is true that any standard programming language can implement the capabilities required for decision support. However, there can be substantial differences in the support that various languages, tools, and development environments provide to support the development, debugging, and long-term maintenance of particular types of large software systems. Based on our experience developing knowledge-intensive intelligent agents, there are a number of capabilities to consider that directly support the development of intelligent systems with significant relational knowledge bases that interact in real time with humans and the environment. For the purposes of this discussion, we generally conceive an intelligent agent to consist of an *architecture* (which provides the primitive support for the general cognitive capabilities that the decision support agent will use) and a *knowledge base* (which provides the domain- and context-dependent expression of the cognitive capabilities for a particular set of tasks). This section describes each of these aspects of the DSA in turn.

5.1.1 Intelligent agent architecture

Before designing the knowledge and behaviors of the DSA, we must select a platform for the agent's design and execution. Traditionally, this would be a *cognitive* or *intelligent agent* architecture. There exist a number of potentially viable architectures that merit consideration as the base platform for the ITATSS DSA. We do not intend to commit to a particular choice in this report. Instead, based on an analysis of the ITATSS system's tasks, we have developed a list of criteria along which potential architectures should be evaluated. In the Phase II effort, we will use these criteria as metric features feeding into the evaluation process described in Section 2.3. The remainder of this section discusses in more detail each of the following important criteria for the ITATSS DSA's architecture:

- Pattern-driven execution
- Support for relational and spatial representations of situational awareness
- Agent development tools
- Efficient implementation for real-time agent execution
- Explicit support for truth and belief maintenance functions
- Support for mixed-initiative execution of goal-driven and opportunistic behavior
- Support for efficient development and execution of very large knowledge bases
- Transparent symbolic representations of knowledge
- Explanation and visualization facilities
- Affordability

One of the most important aspects of an intelligent agent is that it not get locked into scripted action. On a related note, when implementing some course of action or decision making, the agent should be able to handle flexibly interruptions, resumptions, changes in goals, and opportunistically reacting to unanticipated situational changes. All of these issues argue against standard procedural programming, or even script- or plan-based solutions. Rather, the implementation of all long-term decision-making and action should involve encoding the principles and understanding underlying the appropriateness of each discrete action. With such an encoding, atomic behaviors trigger based on a conditional pattern match to the current situation representation. It is also important that the pattern-matching engine support the efficient matching of relational patterns, because of the requirements for relational representation of situational awareness. Additionally, for the types of real-time decision support that ITATSS must provide, it is essential to select an architecture that delivers an efficient implementation of pattern-driven execution.

As with human reasoning, in order for an agent to make the most intelligent decisions, it must first generate a reasonable understanding and awareness of its situation. We should note that this requirement is in contrast to some approaches to intelligent systems and robotics that attempt to create behavior-based systems (Brooks, 1987). While reactive levels of behavior are certainly important, and will be included in our system, such systems create no persistent internal representations of situational awareness, instead encoding purely reactive primitive behaviors. In our experience, however, for human-like decision making and reasoning, an internal representation of state is essential. At least half of the knowledge implemented into the decision support agent will be involved in interpreting the situation and creating a structured representation of the understanding gleaned from these interpretations (Jones et al., 1999). This "belief structure" will then guide the agent's decision-making, goal selection, courses of action, and external behavior. It is also essential that the language provided by the architecture for representation awareness support descriptions of relational and spatial information. Many software applications make simplifying assumptions for the sake of efficiency and software maintainability, coercing situational representations into a propositional framework. However, especially for large systems, this ultimately leads to inefficiency and inflexibility in terms of applying knowledge to the types of situations encountered in strategic and tactical situations. Spatial knowledge will be particularly important for the navigation of robot systems in ITATSS.

Whichever platform is selected for the decision support agent, we can expect the ultimate implementation of the agent to be a fairly large knowledge-based system. This implies that the system will be subject to the normal demands and issues associate with large, long-lived software systems, including initial development costs, debugging costs, and long-term maintenance and upgrade costs. These costs will only be manageable to the extent that there exist reasonable development and maintenance tools for software engineers and subject matter experts involved in the system's design and implementation. There are mature and robust development environments for a variety of programming paradigms, but for some of the more advanced intelligent systems architectures, finding appropriate tools may be an issue.

It has traditionally been an assumption in artificial intelligence that capability supersedes performance. This makes sense, because the primary distinguishing feature of intelligent software lay in its capabilities and competence. However, it could reasonably be argued that the neglect of software efficiency is on of the biggest impediments to the mainstream adoption and deployment of intelligent systems. An essential aspect of most intelligent systems is that they interact well with humans. Interacting well means responding in a timely manner to queries, as well as to changes in the environment. Thus, in providing the other capabilities we address here, we must also keep an eye on their efficient implementation. Particularly for a system that encodes a large amount of decision-making knowledge, the development platform must provide algorithms that will allow that knowledge to be brought to bear on decision support in a timely manner.

Given the necessarily intricate representations of situational awareness, it is important for the decision support agent to maintain consistency across its situation interpretations, as well as their connections to the systems goals, perceptions, and actions. Even with attentive care from a software engineer, it is difficult to build systems that are guaranteed to be self-consistent. Incorporating a truth-maintenance component can be essential to accomplish this. Truth maintenance systems allow the expression of knowledge and constraints in a logical notation. These representations then propagate the entailments of situational updates throughout the situational awareness representation, goal representations, and other aspects of the decision support agent's internal state. The automatic propagation of constraints guarantees consistency and frees the software developer from needing to implement ad hoc methods of creating consistent assertions and cleaning up stale and irrelevant representations.

The requirement for mixed goal-driven and opportunistic behavior is related to the issues of situational awareness and relational pattern-driven behavior. The point here is that intelligence is usually measured quite subjectively as "doing just the right thing at the right time". Often this can mean establishing persistent goals and sticking with them in the face of changing situations. But often the "smart" thing to do is to adapt (and possibly abandon) goals and actions as the situation changes. The key lay in having the agent appropriately "know" when to make strong commitments, when to relax those commitments, and when to take opportunistic advantage of serendipitous changes to the situation. Much of this capability comes from the knowledge encoded in the system, but to maximize the flexibility of the agent, the underlying execution platform must include explicit support for goal driven behavior, reactive and opportunistic behavior, and the continuum of levels of commitment that provide the "most intelligent" responses between these extremes.

Reemphasizing the need for efficiency and development tools, we must accept that a decision support agent with the capabilities desired for an application like ITATSS will necessarily include a significantly large knowledge base. Knowledge-intensive agents are uncommon but not unheard of. Thus, the technology exists to develop them, but care must be invested in selecting execution paradigms that will scale to very large systems. There are a number of intelligent agent frameworks and development tools that have been demonstrated for fairly small agent applications. Almost always, the design assumptions that work well for small systems do not scale to systems with significant amounts of knowledge and intelligence. An essential selection criterion for the decision support agent's implementation platform should be that is has demonstrated the capability of cost-effective development and efficient execution even for agents with very large knowledge bases.

ITATTS is also intended to provide clear, explainable, and transparent decision support and training to a human operator. Based on this fact, our recommendation is to avoid systems that rely on impenetrable representations and black-box processes. To select a straw-man example, the connectionist approach to intelligent systems may not be appropriate for such an application. Connectionism has some significant strengths in terms of providing adaptive systems that tolerate noise and handle flexible representations of knowledge. However, the knowledge embedded in a neural network will not generally be easily explainable, or provide the transparency that would be desired for pedagogical purposes in a training system. For this application, it would be preferable to use a platform that supports symbolic representations using the concepts, goals, and terms that the human operator is familiar with, together with formal and traceable reasoning regimes that are conducive to explaining the decisions recommended by the decision support agent.

Explanatory capabilities are also key to the ITATSS application. The reasons for this are twofold. First, when used as a training system, it is essential that the decision support agent be able to explain its decisions and actions in a natural and understandable way to the human operator. But even in an eventual operational deployment, explanatory capabilities would be important. Human operators are (appropriately) skeptical of decision support systems and intelligent systems in general, and want to know the source and justification of any recommended courses of action (Barnett and Meliza, 2003). A system that can reasonably explain its actions will receive much broader acceptance in the user community. Additionally, explanation need not and should not come only via verbal summaries of decision-making processes. Multi-modal explanation systems would include text, graphical visualization, and possibly other modes of communication. The ideal candidate for developing the decision support agent would include integrated tools or frameworks for generating and presenting explanations in a multi-modal fashion, particularly with annotated graphical representations in a user-friendly interface.

Finally, we must consider architecture costs, particular if we consider using COTS systems for the agent architecture. We have mentioned cost as an issue in some of the other requirement descriptions, but we reiterate those concerns here. It must be accepted that a robust, intelligent, decision-support and training system will be somewhat expensive. However there are areas in which costs can be controlled. As we have mentioned, it is desirable to have development and maintenance tools that can decrease engineering costs. When selecting an implementation platform for the decision support agent, there are other potentially recurring costs having to do with commercialization, licensing, and deployment. For the set of candidate implementation platforms, there are a variety of accessibility paradigms, from public domain software to open-source licenses to fully restricted (and often expensive) development and deployment licenses. The potential recurring costs of using licensed software should also be evaluated when selecting an implementation platform.

5.1.1.1 Example candidate agent architectures

There are a variety of classes of development platforms and architectures that should be considered and evaluated along the above dimensions before making a final selection for development of the decision support agent. At one end of the spectrum are what we might call "traditional" software development languages and environments, using standard programming languages such as Java or C++. The major strengths of this class of platforms are that they are widely available, widely known by software engineers (and thus there will never be difficulty finding engineers to develop and maintain the system), and there are sophisticated development and bug-tracking tools available for these platforms. On the downside, this class of solutions does not directly support many of the programming idioms and patterns described above, thus requiring ad hoc solutions for their implementation in the ITATSS system. This could have an impact on system efficiency, as well as long-term software maintenance costs. One popular approach to building constructive forces using standard software development loosely relies on the notion of finite state machines. This is an attempt to maintain complex software manageability while also providing an efficient system that has the advantages of traditional software engineering. There are also some significant weaknesses to this approach, but with some possible adaptations, the approach should be considered as a candidate for evaluation in ITATSS.

At the other end of the spectrum is the set of mature cognitive architectures that have been developed over the past 30 years with the explicit intent of building systems with human-like intelligence. Some of the most prominent examples include ACT-R, Soar, Epic, and Glean. While each architecture has its individual strengths and weaknesses, they each generally provide primitive support for many of the design capabilities that we have identified as essential for the development of intelligent systems. The risks involved in using such systems is that they are not all robustly engineered, they have limited communities of engineers familiar with them, and the development and maintenance tools associated with them are generally not as mature as for traditional programming environments. However, the capabilities these systems do provide are significant enough that they should receive significant attention in any evaluation for development of the ITATSS decision support agent.

Between these two extremes are emerging software development platforms that attempt to combine the strengths of standard software engineering approaches with primitives that support the development of intelligent systems. We can roughly label this class of systems as "agent-oriented" development platforms that take engineering concerns seriously, but are not full-blown cognitive architectures. The most prominent examples of this class include implementations of the Beliefs-Desires-Intentions (BDI) framework for agent systems, such as JAM and JACK. Such development systems should also be considered for evaluation, because they attempt to make pragmatic tradeoffs between the competing concerns for the development of intelligent agents. Perhaps the most significant concern for these types of platforms is that they are not quite as mature as some of the other approaches and have generally not been proven on the deployment of significantly large agent systems.

5.1.2 Knowledge-based agent for decision support

Having outlined requirements and metrics for selecting an agent architecture, we move on to discussion of requirements and design processes for the DSA's knowledge base. At a high level of description, there are a handful of requirements for the performance of the decision support agent. However, as with most systems, and certainly with intelligent software systems, the devil is in the details. In support of the former, we will begin with a brief overview of these high-level capabilities. The remainder of this section will examine in more detail some of the particular requirements and complexities embedded within each high-level task.

5.1.2.1 High-level capabilities requirements

As we have mentioned previously, our experience demonstrates that knowledge-intensive agents can be characterized by a high-level organization that comprises three basic types of knowledge. This section describes each of these knowledge types.

5.1.2.1.1 Knowledge for situation interpretation

Situation interpretation knowledge provides the basic language and terms that experts use to formulate and reason about their task. This is the language from which complex representations of situational awareness can be created. For tasks requiring intelligent thought, there is no hope of generating appropriate and intelligent behavior unless the agent can first create an accurate "picture" and assessment of its situation. This representation language begins with the primitive types of sensory information provided by the environment to the DSA. In the ITATSS architecture, "primitive" sensory information will consist of symbolic structures that have been generated by the early processing filters in the system. Early processing will include, for example, vision systems on the robots, monitors on the robot's motor systems, and terrain analysis preprocessors. Additionally, there will be higher-level types of "sensory" information from external sources, such as communicated directives and requests from humans or other systems, inputs from the human operator, and symbolically structured mission and intelligence information gathered from other digital sources.

The DSA's situation interpretation knowledge integrates the sensory information into coherent structured representations. Importantly, it also re-represents the information using higher level labels and descriptions, in a manner similar to a human expert. As a simple example, when a human expert detects a potential moving threat with a specific sensed heading and velocity, an expert can further massage the sensory information into higher level "features" that are useful for subsequent reasoning, such as "the threat is pointing at me" or "the threat is closing rapidly/slowly". Most of the situation interpretation knowledge accomplishes this type of translation from primitive sensory features into higher-level terms that directly drive action and achievement of goals.

In the UGV domains in which the ITATSS DSA will operate, example higher level representational features would include things like routes, obstacles, an ontology of high-level terrain features (e.g., buildings, hills, forests, cities), important locations and landmarks, descriptors for enemy locations and activity, descriptors for friendly and neutral locations and activity, weather, mission payload capabilities, enhanced digital map features, satellite imagery assessments, and useful representations of the passage of time and action coordinators.

5.1.2.1.2 Knowledge for purposeful reasoning and behavior

An intelligent agent must appropriately commit to courses of action, knowing when to maintain particular plans, and when to abandon them or reactively adapt them to changing situations. This requirement for intelligent commitment implies some sort of representation of purpose or goals. From a practical perspective it is also often useful for intelligent agents (as well as human experts) to be able to compartmentalize their decision-making and response into a variety of goals and subgoals. Such compartmentalization allows a system (or human) to minimize cognitive overload while focusing attention on "reasonably sized" portions of the task, and also enhances the possibility of reusing or adapting knowledge "units" appropriate to one type of goal, so that can be used for similar types of goals. All of this ordinarily implies some sort of goal/task hierarchy, allowing the decision support agent to decompose high level directives into refined components for which it has deep, context-sensitive knowledge. However, in complex real-world domains, we are often faced with situations where goals and subgoals do not fall into a clean hierarchy. Rather, goals interact with each other, hinder and/or help each other, and often must be pursued in parallel and/or traded off with each other. In military domains, the top-level goals often arrive in the form of a mission briefing. Mission goals often must be integrated with (potentially conflicting or interacting) strategic goals, rules of engagement (which are also types of goals), lower-level tactical goals, and goals that may be derived from a unit's standard operating procedures. Additionally, all of these goals must integrate into a system that includes more persistent "universal" types of goals, such as the goal to survive, protect the payload, minimize costs, etc.

We have found the best representation for complex interacting goal structures is a dynamically self-reconfiguring directed acyclic graph (DAG) composed of sets of interacting hierarchies. For the ITATSS DSA, some of these hierarchies will represent the persistent sorts of goals, such as "survival" (or maintaining the integrity of the physical platform) and force protection. There will also be a fair number of mission-level goals. For our design and demonstration scenario, potential mission-level goals will include reconnaissance goals (further refined into subgoals such as route survey and classification, location of enemy, and point/area/zone reconnaissance), surveillance (further refined into subgoals for achieving observation and/or listening points), communications retransmission, demolition (of weapons or enemy targets), or forced entry (by blast or planted explosive). All of these mission level goals will share subgoals, particularly having to do with navigation, route following, and maneuvering, but also in terms of avoiding detection, delivering payloads, maintaining communication channels, etc.

5.1.2.1.3 Knowledge for external action

Ultimately the decision support agent must know how to perform its task. In the long term, the ITATSS DSA must know how to control an unmanned ground vehicle to accomplish the defined missions and maneuvers. In the initial vision, the DSA does not need detailed knowledge of how to actually realize this control, because the initial application of the system is for support of a human decision maker. Thus, rather than needing to know how to move a joystick to perform a particular robot maneuver, the DSA only needs to be able to form and reason about the *intentions* to perform low-level actions. At that point, the DSA can communicate possible intentions and ramifications of those intentions to the human operator. However, this still leaves a lot that the DSA must know. While it does not need to know exactly how to perform particular maneuvers, it must understand the strengths and limitations of each possible external actions, plus it must have some representation of how each action will change the robot's situation (feeding into the situation interpretation knowledge base).

The types of external action that the DSA must know about are limited only by the capabilities of the robotic platforms. The decision support agent must have the knowledge to reason about any type of action that is relevant to the overall mission, and that has an impact on the decision-making that the human operator must perform. For our demonstration scenario, external action knowledge would include (but not necessarily be limited to) the maneuvering controls on the specific robotic platform, external control of sensory systems, radio or other communications device handling, weapons deployment or employment, and control of navigation systems.

5.1.2.2 Details of decision support

The ITATSS DSA must supplement its knowledge of how to *perform* the robot operation task with knowledge of how to *support* a human performing the task. In order to maximize effectiveness, a decision support system should ultimately make it easier for a human operator to learn and/or perform their job. Accomplishing this requires building in an understanding of what that job is and where support can be most effective, exploiting the strengths of the computational system to target specifically those aspects of the task that are difficult or unfamiliar to the operator. The best approach to characterize the task is with a cognitive task analysis (Card, Moran, & Newell,

1983) involving a user-centered approach that relies on real SMEs who are familiar with the task and its areas of difficulties.

Once complete, the cognitive task analysis serves as a detailed template for requirements on the behavior of the DSA. Given the nature and current state of the art of intelligent systems, there is rarely a point at which the design and implementation of the decision support system could be called "complete"; there is always room for improvement or refinement of the knowledge and intelligence embedded within the system. This fact demands a spiral design approach, with the initial support system providing fairly broad but somewhat shallow coverage of the required tasks. The iterative design process then works the highest priority portions of the task, to provide deeper levels of understanding and intelligence to the support system, always with a view for future extension of the system's capabilities. The spiral design approach is also heavily user centered. Each iteration of the system provides an interactive platform for further requirements definition, where subject matter experts can clarify and add to the desired capabilities for the system.

For the initial design effort, we have interviewed a subject matter expert to determine an initial high-level task analysis, together with a decision support analysis that identifies key decision points in our design scenario and suggests useful actions that the DSA could provide at each point. We have included the basic initial requirements on the DSA in the sections of this report that detail the DSA's interaction with the various user interfaces (Section 4) and functional components (Section 5).

5.1.2.3 DSA behavior development plan

As mentioned above, a useful approach to development of decision support agent behaviors is a spiral design, with heavy involvement of subject-matter experts and users in the requirements, prototype, and test phases. This is in large part due to the inherently complex nature of intelligent systems, and the ill-defined requirements that are part and parcel of building an "intelligent" system. There is always room for an intelligent system to be even more intelligent, and (as is also the case with standard, large software systems) there can be unlimited refinement of the requirements for such as system, as it sees increasing use in a particular application.

These facts point to a design approach that begins with a broadly focused specification of knowledge for the decision support agent, with appropriately deeper requirements specified for initial prototype design and testing scenarios. We therefore begin the process as above, with an initial high-level cognitive task analysis, together with a user-oriented study that highlights which portions of the system's behavior should initially be emphasized. The cognitive task analysis formally specifies the high-level goals that that system will be called upon to reason about, together with an initial pass at the representation of situation-interpretation knowledge and the set of external behaviors that should be generated by the system. The usability analysis tells us which parts of the goal and situation representations to elaborate in the construction of the first prototype system.

The design and implementation of the initial prototype is further guided by an "augmented" prototype scenario, or set of scenarios. The idea of the prototype scenario is to provide a specific focus to the development of the prototype agent, but to be general (or generalizable) enough that the scenario broadly encompasses a large number of the subgoals and subtasks that the decision support agent will ultimately need to implement in a more complete manner. The resulting agent design should provide a decision support agent that generates reasonably appropriate behavior for the high-level decision points in the prototype scenario.

After construction of the prototype decision support agent, the spiral development process begins in earnest. Using the prototype agent in the prototype demonstration and testing scenario, the subject-matter experts and users, together with knowledge engineers, can evaluate the implementation of the initial requirements, and further refine the requirements for the next iteration of the spiral. User-centered development and testing of the prototype are key here. The prototype scenario provides solid context for the decision-making of the decision support agent, as well as the evaluation of that decision-making by the subject-matter experts. It is very difficult to extract and engineer such knowledge from the experts without this kind of rich context.

However, it is also important to design easy extensibility into the system from the start, because it is difficult or even impossible to know ahead of time which aspects of the system will merit the most attention or be the most difficult to engineer outside of the context of the prototype demonstration and testing.

An additional key aspect of our approach to decision support agent implementation is to encode redundant explanatory knowledge along with the performance knowledge of the decision support agent. Explanatory knowledge serves in part as a "semi-neutral" knowledge representation that defines the requirements and content of the system's knowledge without involving the idiosyncrasies of a particular performance engine (where the performance knowledge necessarily includes engine-specific idioms and implementation). But the explanatory knowledge is also invaluable for system testing and requirements refinement. With the additional of explanatory knowledge, the decision support agent can explain, in human-friendly terms, the reasoning behind any of its decisions. We have developed separately an engine-neutral framework for constructing and presenting such explanations (Taylor et al., 2002). Figure 24 shows the basic relationships between components in the VISTA framework. The same knowledge base can be compiled both to perform a task and to explain the decision being made by the performance agent. The VISTA display presents an integrated view of the performing agent's situational awareness and explanation agent's explanatory annotations. This type of explanatory capability aids spiral development by allowing the involvement of the subject-matter expert into the debugging and requirement refinement process, because system behavior is presented in the subject-matter expert's language instead of the knowledge engineer's language.

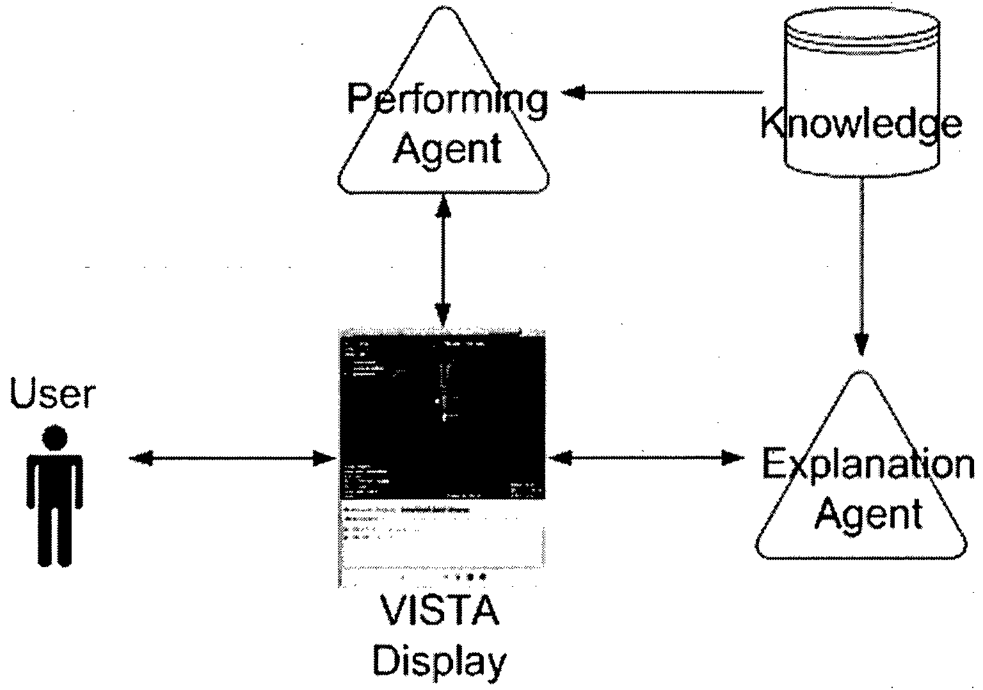


Figure 24. Using redundant knowledge representations, we can automatically generate performance and explanation agents that interact with VISTA's displays of situational awareness.

5.2 Terrain analysis/update via robotic control

It is important to be able to provide terrain models at multiple levels of representational access to the user for the control and validation of both actual and simulated missions. This is important due to the unpredictable nature of the missions requested and the run-time conditions encountered. Fortunately, MissionLab already possesses the capability to incorporate digital terrain models. This work has been used previously for elevation data to test communication models as part of DARPA MARS Vision 2020 program using models provided from our colleagues at the Fort Benning McKenna MOUT facility. What we would like to be able to do is to expand this work toward the following ends:

- Incorporate high-fidelity environmental and robotic feedback into both simulated and actual missions for the robot operator to more effectively validate mission intent prior to departure and enhance control.
- Provide a mixed force-on-force operating environment based on an integrated MissionLab-OTB environment for testing using both real and simulated entities concurrently (Drewes, 2003).
- Use actively sensed terrain data to configure control parameters during execution in order to dynamically ensure vehicle safety in the presence of a poorly trained, stressed, or otherwise careless operator (Castelnovi, Arkin, and Collins, 2005).

MissionLab has been proven to be highly flexible in its design, and indeed it has already been successfully ported to multiple architectures, simulators, and robots. These architectures and simulators include ROCI, Player/Stage, MRPL and AuRA, and as we have considerable experience in this area we foresee little difficulty in porting this work to OTB. JAUS compliance is also a potential extension easily within reach. Georgia Tech is currently working with SAIC-Orlando on an IRAD project (Drewes, 2003) where Georgia Tech software has already been integrated into their OTB-robot system. Thus a feasibility proof exists for the compatibility of our work efforts. This system includes a mix of virtual and real robotic platforms, which is also suitable for training and exercises.

Earlier work at ARL (Fields, 1999), including documented software function calls that permit the modification of the Terrain databases for UGVs in tactical scenarios using ModSAF can quickly provide an integration pathway. One of the larger issues deals with microterrain and simulation time-step size, as opposed to the existing coarse resolution in both time and space, that will be needed for a range of experiments and field exercises but candidate methods have been developed to address these issues (Fields, 2001).

Finally we have recently developed (Castelnovi, Arkin, and Collins, 2005) a novel method for real-time terrain characterization using a Segway RMP platform with a downward looking SICK laser scanner (Figure 25).

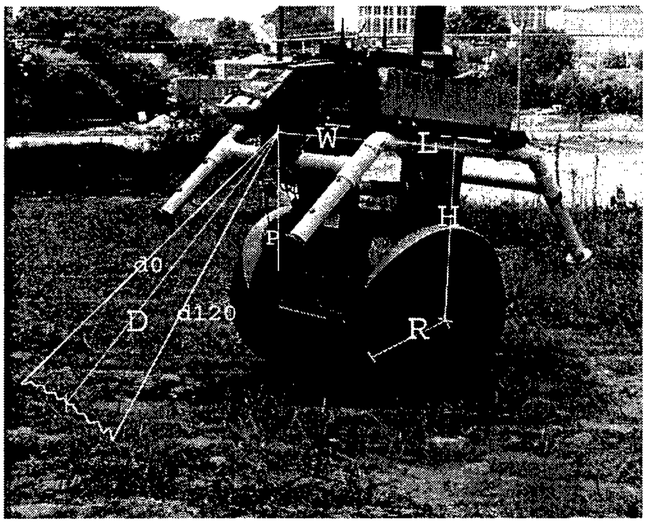


Figure 25. Geometry of a SICK on the Georgia Tech RMP. (See Castelnovi, Arkin, and Collins, 2005 for additional details.)

The resulting terrain data can be entered into existing terrain maps to provide both microterrain models and updates on the quality of the existing map data. The current intent of this effort is to ensure overall safety of operation during teloperated control, by managing the maximum velocities that the robot can travel based on computed terrain classification. As MissionLab already possesses the ability to represent 3D terrain elevation models, these extensions should be straightforward, relying predominantly on converting to new formats (e.g., DEM) and conventions as required by the application.

5.3 Feature analysis

ITATSS will use a path planning capability developed for the DARPA MARS and PerceptOR programs. Stentz' D* path planner is used for both path planning on prior map data and dynamic replanning based on incremental updates provided from sensor data. D* is similar to the classic A* state space planning algorithm, but provides guaranteed optimal paths with minimal recalculation in the presence of changing state costs. Figure 26 below shows path planning based on a cost map derived from elevation data. The left image shows the terrain cost map, with darker areas representing higher cost. A user has selected two waypoints, shown connected by the straight yellow line. The end point lies inside a triangular fortification with a narrow opening. The image on the right shows the planned optimal path, which negotiates the obstacles visible in the elevation data and enters through the narrow entrance.

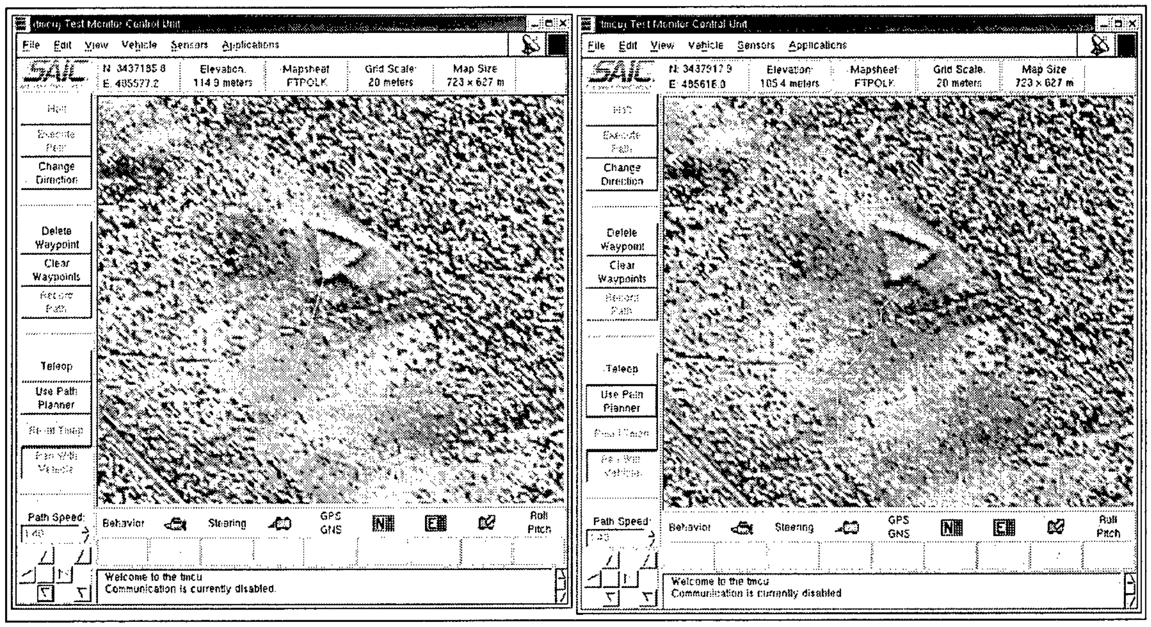


Figure 26. Premission D* planning on a terrain cost map derived from elevation data.

Cost data collected by the robot during mission execution is merged with prior data to support efficient dynamic replanning using the D* algorithm. Data from the local vehicle-centered terrain map kept by the robot is transferred to the global map as it passes outside the bounds of the local map. Figure 27 below shows an example of cul-de-sac recovery during mission execution using this dynamic replanning capability. The red path shows the straight-line path between mission waypoints. The shades of brown represent traversability cost. The dark region in front of the orange vehicle icon represents an untraversable region detected by the vehicle sensors during mission execution that is blocking vehicle progress. The blue path shows the results of replanning, which provides a way to escape from the cul-de-sac and reach the next waypoint.

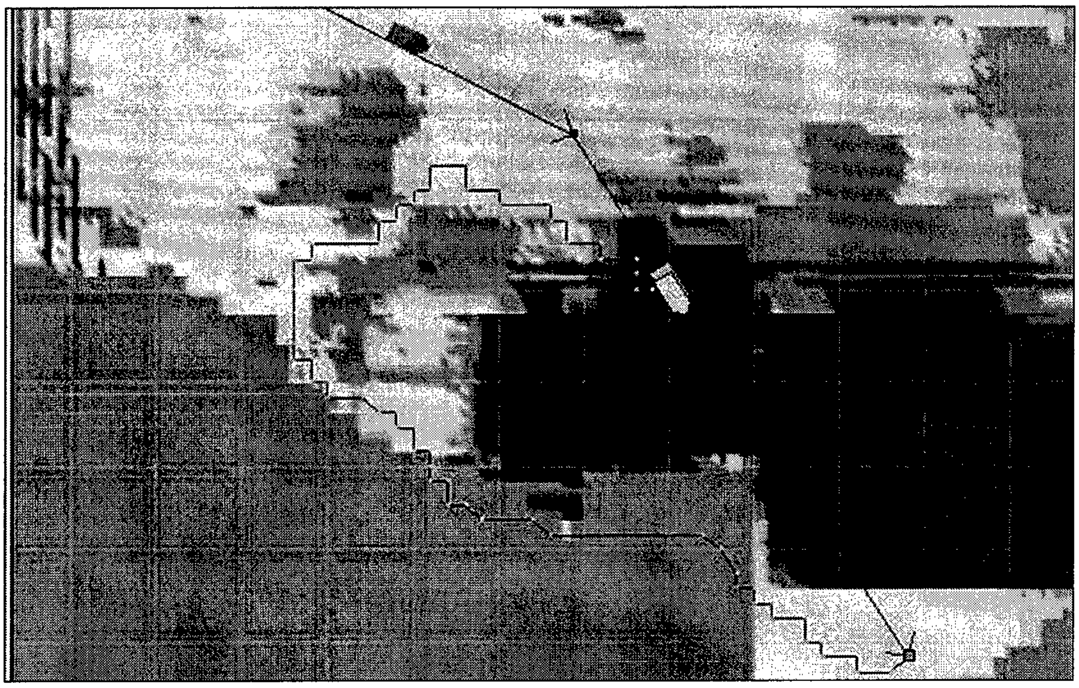


Figure 27. Dplanner cost map and planning results showing cul-de-sac recover from an all blocked state.

At any give time traversability cost can be computed based upon the current state of the dynamic terrain classification model. Cost queries are supported for points, lines or regions. Figure 28 illustrates the concept of dynamic terrain classification features and traversability cost map calculations in the case of a route planner that uses a pre-computed cost map to perform route planning. In the context of PerceptOR, the cost map was derived solely from mobility considerations, however the infrastructure for incrementally integrating new cost information into the global map and dynamically replanning vehicle paths is independent of the source of the cost values.

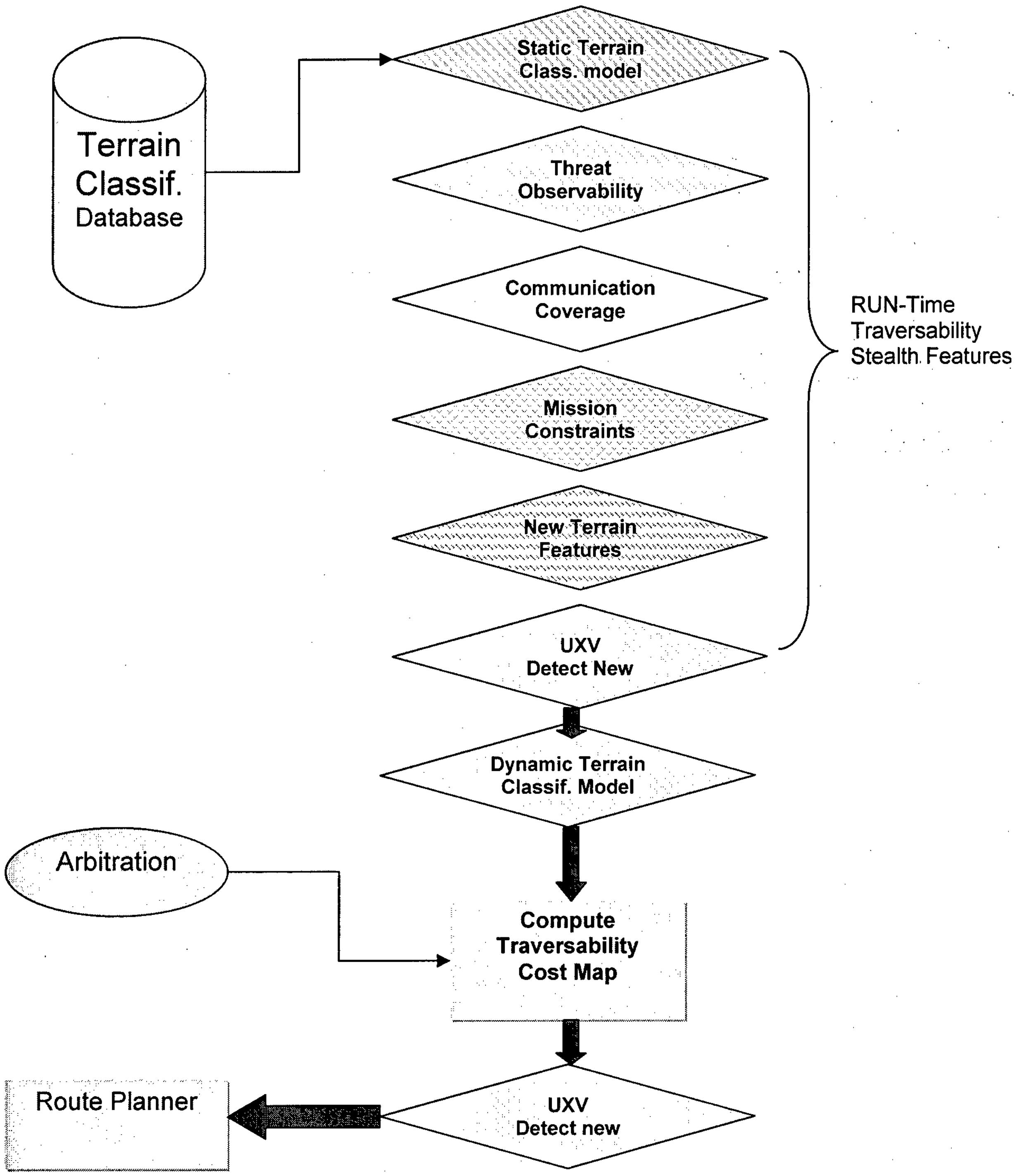

Terrain Classif. Database
Static Terrain Class. model
Threat Observability
Communication Coverage
Mission Constraints
New Terrain Features
UXV Detect New
RUN-Time Traversability Stealth Features
Dynamic Terrain Classif. Model
Arbitration
Compute Traversability Cost Map
UXV Detect new
Route Planner

Figure 28. Elements of the static terrain classification map.

Demo III software configuration for the Planning Services diagram is illustrated in Figure 29. There are five configuration items: 1) World Model Management, 2) Route Planning, 3) Visibility & Threat Analysis, 4) Communications analysis and 5) Tactical & Emplacement Reasoning.

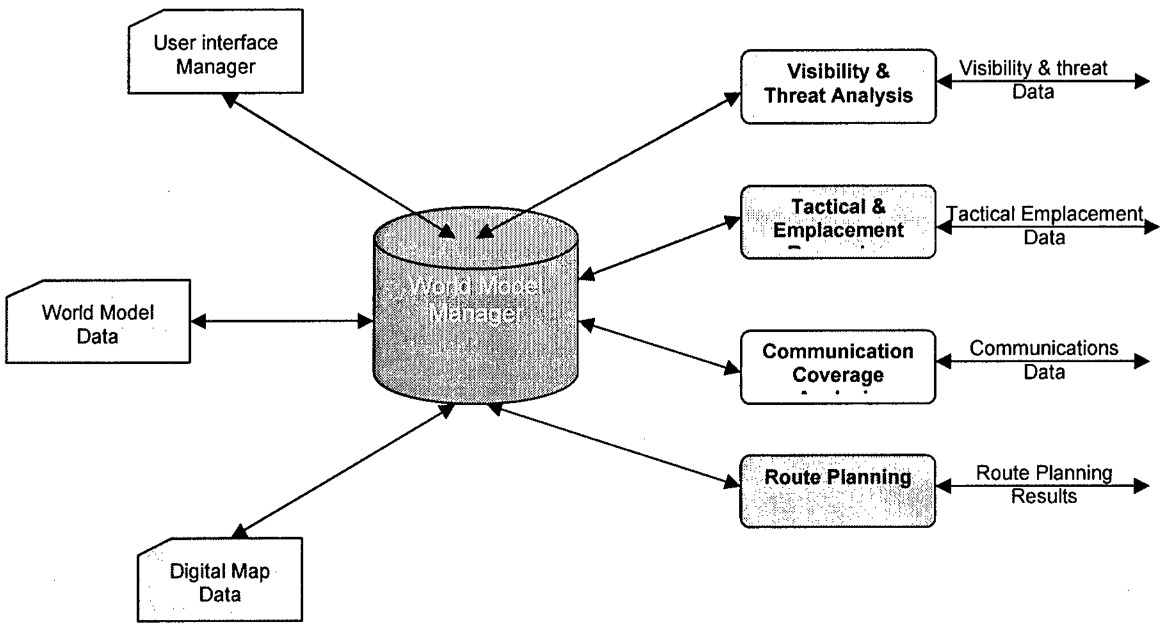


Figure 29. Planning services software configuration.

5.4 Situational event detection

In addition to the perceptual algorithms required for supporting navigational requirements and standard situational awareness, additional perceptual event monitors will be created and instantiated on a mission-specific manner. These algorithms will trigger an alarm to both the operator and the decision support agent for the need to possibly replan or suspend the mission in light of incoming field information provided in situ by the robot. A few simple examples include:

- Loud sound detector, possibly indicative of incoming artillery
- Person detection, possibly indicating the need for invoking stealth operations
- Vehicle detection, similar to person detection
- Expected terrain violations, where major disagreements exist between the terrain database and the robot's observations.
- Others on an as-needed and mission-specific basis.

In our initial design for ITATSS, simple perceptual algorithms, such as color blob detection will serve as placeholders for more complex algorithms to be developed in other programs.

5.5 Robot action decoder

The Robot Action decoder, as illustrated in the ITATSS Notional architecture diagram, serves as the bridge between the decision support agent and the executing robot plan.

It is an important and crucial link between these two components. It is one of the pieces of ITATSS that is not off the shelf and will need to be developed specifically for this effort. It serves to tie together the decision support agent's decision output and the MissionLab run-time execution system.

The requirements for the robot action decoder are:

1. Translate high-level robot commands into a form compatible with the robot sensorimotor interface, in particular, the proposed CDL/CNL and HServer run-time components derived from MissionLab.

2. Preserve the semantics of the command stream from the decision support agent all the way through actual robot execution.
3. Efficient and compact data structures to ensure real-time performance.
4. The ability to be generalized to multiple robot systems in future work, thus including vehicle/subteam/team IDs to be associated with commands.
5. Similarly allow a robot architecture independent format to ensure generalizability to other systems (e.g., JAUS/4D-RCS).
6. Adequately represent the output of the decision support agent (completeness).
7. Execution checking to ensure that the commands produced by the DSA are still timely and relevant prior to their execution.

Interface Design aspects for the robot action decoder include:

1. Production of a command language shared between the DSA and abstract sensorimotor interface that adequately represents the scope of commands to be generated via the decision support agent. This syntax will be produced in a readily extensible manner so that the architecture can be easily migrated to new task domains as required by the Army.
2. This new Interlingua will require suitable translators/filters from the existing SOAR outputs to this new language and similarly a command translator into a CDL/CNL format of appropriate use by the abstract robot sensorimotor interface.
3. Adding precondition checking to ensure that commands that are sent from both the operator and DSA are still valid and executable in the actual context that the robot currently resides within.
4. Rigorous documentation of the interface and command language structure to ensure feedback.

Our extensive experience in building languages of this sort already, i.e., Configuration description language (CDL), Command Mission Definition Language (CDML) and the configuration network language (CNL), should make this a straightforward task.

It is necessary, in order to be able to support a broad range of heterogeneous robotic platforms and potentially multiple architectures to design an abstract interface that connects the planning and control activities with the actual robotic hardware. Several significant desired characteristics include:

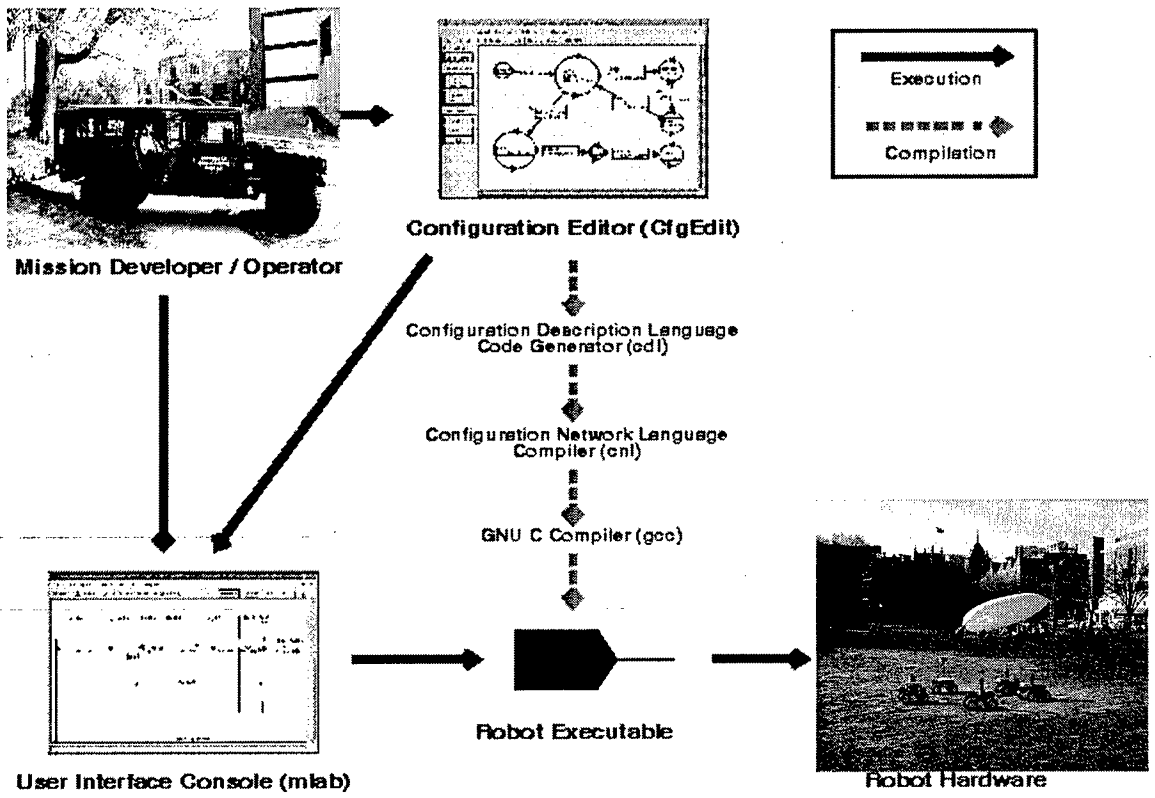
- Support of virtual sensors and actuators
- Isolation of hardware dependencies from remainder of architecture
- Modular design with reusable components
- Efficient run-time communications of both command and sensor data.
- Communication with simulated platforms in identical manner as actual vehicles.
- Reliable performance through the use of run-time monitors
- Formal verifiability of properties, e.g., guaranteed delivery, deadlock avoidance

Historically, most systems to date have been designed in a customized ad hoc manner, with little concern for generality. There are two notable exceptions. Player/Stage (Gerkey, Vaughan, & Howard, 2003), developed at the University of Southern California, and MissionLab (MacKenzie, Arkin, & Cameron, 1997), developed at the Georgia Institute of Technology. While both of these systems have their advantages and disadvantages, (e.g., Player/Stage is Open Source with a large user community contributing to its development), MissionLab is considered more appropriate for this particular task for a variety of reasons. These include:

- A larger scope than Player/Stage, focusing on end-to-end mission specification to actual low-level control
- MissionLab is a superset of Player/Stage functionality. Indeed compilers exist for Player/Stage within MissionLab developed as a part of the DARPA MARS 2020 program.
- MissionLab readily supports multiple robotic architectures, including AuRA, MRPL, Player/Stage, and ROCI. New approaches including JAUS compatibility should be easily attainable within this system.
- Both MissionLab and Player/Stage, supports multiple ground robotic platforms, but MissionLab has been ported to a wider range of vehicles from small amigobots to a Commercial Hummer H-1.
- MissionLab has from its very inception, unlike Player/Stage, been designed as a hybrid reactive/deliberative architecture, able to integrate spatial planners into its design.
- MissionLab has been under development longer (since 1994) and has been supported under 7 different DARPA programs.

For these and other reasons, MissionLab serves as the point of departure for the design of the sensorimotor interface for ITATSS. The system is actually a suite of tools ranging from a mission specification visual programming GUI to low-level suites of reactive behaviors, to interprocess communications software (IPT provided by CMU), to hardware specific translators for robotic platforms.

Abstraction arises from the use of intermediate languages that are generated from the user's


Execution
Compilation
Mission Developer / Operator
Configuration Editor (CfgEdit)
Configuration Description Language Code Generator (cdl)
Configuration Network Language Compiler (cnl)
GNU C Compiler (gcc)
User Interface Console (mlab)
Robot Executable
Robot Hardware

mission intent. The current mission specification process appears in Figure 30.

Figure 30. MissionLab mission specification process.

In particular note the series of translators and compilers that connect the visual mission description to the actual platform. Each of these provides a different level of abstraction. Assuming that the AuRA Architecture [*], developed at Georgia Tech is the target architecture, the following sequence of events occurs in tying the specified mission to the robotic hardware.

1. The graphical description is translated into CDL (Configuration Description Language), and architecture independent language to express complex robotic missions.
2. CDL is then bound to an architecture, in this case AuRA, and the appropriate compiler is invoked. In this case it is CNL, which is the Configuration Network Language.
3. The resulting CNL code is then translated into C++ code and then compiled by a Gnu C++ compiler.
4. The resulting executable is downloaded to the robot for execution.
5. Hserver (Hardware Server) is started on the robot, which translates the appropriate abstract motor commands into robot-specific instructions while also providing the conduit for sensor information back to the behavioral network for processing.
6. IPT (interprocess threads) are then established between the network of behaviors the operator console, Hserver, and other relevant run-time processes.
7. The mission begins and continuous updates of the robot are reported from the robot through Hserver via IPT to the operator console.

Thus whenever adding new sensors or new actuators, the only components within MissionLab that need to be rewritten to command or access the sensor data is Hserver. This isolation of design change enables the system to be readily ported to new platforms, sensors, and even architectures far more easily than any other system that preceded it.

The information provided in the MissionLab documentation (currently in version 6.0), available at http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/, includes the User's manual which contains a description of the CDL language, details on Hserver, and a separate CNL user's manual, This documentation provides the details regarding the specific abstractions used, many of which have their roots in formal language theory (MacKenzie & Arkin, 1993).

For future development considerations beyond the research contained within the early phases of the STTR effort, MissionLab is being commercialized under license from Georgia Tech, by Mobile Intelligence Inc.

5.6 Reuse of existing systems and technologies in ITATSS

A significant advantage of the ITATSS design and development team is that we have extensive experience developing systems that provide many of the functional components of ITATSS. The ability to reuse these systems and technologies will provide considerable leverage in the design and implementation of the new architecture. While it is impossible to give definitive quantitative performance evaluation without actual construction of the overall system, several important feature characteristics can be observed.

- ITATSS will inherit the run-time efficiencies already present within the MissionLab system. The actual robots are controlled by their own mission-derived executables, which are capable of safe performance even in the absence of operator or DSA input.
- The architecture will also incorporate existing tools to generate mobility cost maps based on elevation data, as developed for the Demo III and PerceptOR programs.
- There are also existing modules to support incremental integration of sensor-derived terrain data into a global terrain map, and to dynamically replan vehicle paths based on the updated terrain costs using the D* algorithm, as developed for the DARPA MARS and PerceptOR programs.

- As the DSA acts as a supervisory agent for both the human operator and the robot regarding terrain matters, simple tasks such as adjusting maximum speeds based on velocity can occur through parametric substitution, a practice already in place within the usability-tested MissionLab human-robot interface.
- Currently IPT serves as the real-time multi-threading communications component for MissionLab. It is proposed that this interprocess communication component will continue to be used within the robot interface and exported as needed for communication with the other major inter-component communications of the ITATSS architecture.
- CDL has already been proven to be portable to a broad range of robotic systems, including ATRV-Jrs, Pioneer I and II, Amigobots, GT Hummer, Segway RMP, Evolution Scorpion, Nomad 150 and 200, and Denning Robots. This portability will allow straightforward extension to GFE vehicles with minimal code generation as the Army transitions this research from laboratory to field to demonstration and then beyond.
- A new interpretive version of CDL++ has just been generated and demonstrated successfully at Ft. Benning in December 2004 as part of DARPA's MARS Vision 2020 program. This language allowed for easy portability not only to multiple heterogeneous robots, but also to multiple heterogeneous architectures from disparate sources (CNL/Georgia Tech, ROCI/UPenn, and Player/USC). This ability to retarget missions in a highly abstract way will make it very efficient to port the results of this research to other DoD architectures, e.g., JAUS and 4D/RCS.
- The key timing issue is coordinating the outputs of the DSA in a way that it produces commands that are useful in real-time. Thus new aspects of time stamping and command relevance will need to be introduced to ensure that the results of the deliberative process, which operates on a longer time horizon, are not outdated by the time they are requested to be executed. Thus certain condition checking will need to be added to CDL in order to support this wholly new aspect of agent-based deliberative reasoning being integrated into the robot interface.
- VISTA is an existing Java-based implementation of an infrastructure for visualizing situational awareness elements. VISTA includes an implementation of a shared knowledge repository similar to that in ITATSS, and current work on VISTA is supplementing the system with automated facilities for generating explanations of intelligent agent behaviors.
- The state of the art for knowledge-intensive agent architectures allows the development of large-scale intelligent agents that incorporate large amounts of task knowledge and can interact with human users flexibly and in real-time. As an example, the Soar architecture has been deployed in a variety of military training simulation systems, and there have already been exploratory integrations of Soar with the VISTA visualization framework and the OTB simulation platform.

Summarizing, the design of ITATSS is such that it should readily inherit the proven run-time capabilities and efficiencies of a number of existing systems without difficulty. The task ahead is in defining and implementing the interfaces between these existing systems and the ITATSS shared knowledge repository. Defining these interfaces will include the issue of determining how to effectively integrate deliberative high-level decision making without disrupting reactive control for low-level operations such as obstacle avoidance. The robot action encoder specifically addresses these issues.

6 Training and simulation

A significant milestone leading to Demo Alpha was the integration and test of the OCU with the Mounted Maneuver Battle Lab's virtual simulation environment. This capability, which was jointly developed by the MMBL and the system integration team, enabled the modeling, simulation and experimentation community to conduct a virtual BLWE prior to the Demo Alpha live experiment. In these experiments the Battle Labs conducted numerous trials of force on force scenarios in a battle space comprised of simulated manned forces and the UGVs. The focus of the experiment was to evaluate the OCU and the operational concept it represents for command and control of unmanned scout vehicles as part of a mixed force structure. Additionally, this event served as a venue for previewing the training support package that accompanies the UGV system.

Figure 31 shows a block diagram of the OCU interface to the Distributed Interactive Simulation (DIS) environment. In this setup the OCU software is configured to run on standard commercial off the shelf desktop computer. A commercial version of the OCU touch screen display was integrated with this workstation for added realism. The key-enabling component is the UGV interpreter, which acts as a translator between the UGV system and DIS network. On one side, the UGV interpreter communicated with the OCU using the same command set as the UGV system. On the other it used DIS protocols to communicate with simulations of the UGV and other entities in the experiment. A customized Stealth Viewer, an application capable of generating real-time 3D visualizations of the DIS world, was configured to be able to generate synthetic RSTA imagery similar to that generated by the real UGV system. ModSAF workstations were configured to emulate the UGV behavior and functionality. The UGV interpreter coordinated the activities of these various machines in response to commands received by the OCU and events as they occurred in the simulated battle space.

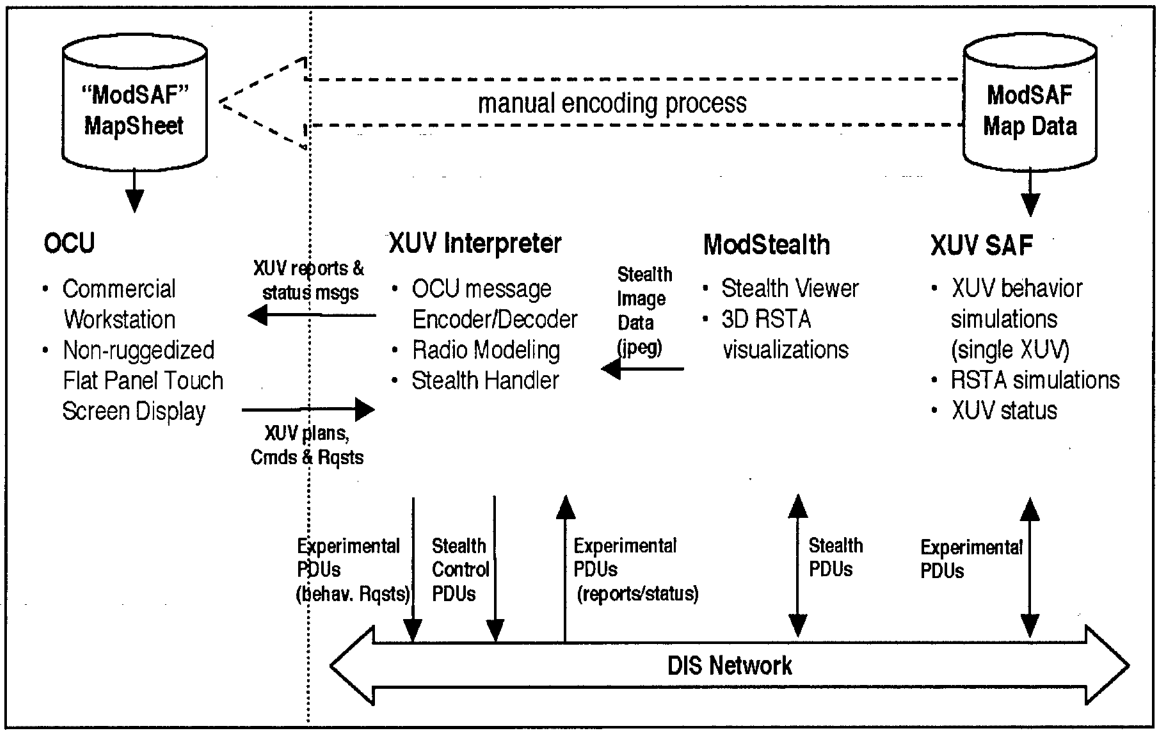


Figure 31. The OCU interface to the DIS networked environment.

The high degree of variability of appearance of relevant types of cross country terrain features, which rarely conform to geometrically simple shapes, results in a degree of brittleness in hand-coded methods for obstacle detection/obstacle avoidance (OD/OA). As a result, current obstacle avoidance methods for autonomous cross-country navigation suffer from performance problems due to both false positive obstacle detections (traversable terrain such as sparse brush being classified as obstacles) and false negative obstacle detections (fallen logs capable of stopping or high-centering the robot not being classified as obstacles). In the case of false positives, progress towards the robot's goal is slowed due to added path length from unnecessary avoidance maneuvers. In the case of false negatives, progress is slowed both by added path length from backing, and by time taken for operator interventions to stop and recover from repeated attempts to pass through the same obstacle. In addition, the use of hand-coded methods of terrain analysis tends to lead to the proliferation of special cases and algorithm parameters in the system. This results in decreased software reliability and maintainability. The application of learning methods to the OD/OA problem offers the potential to address the problems described above: to increase the robot's rate of progress towards its goals by reducing unnecessary maneuvers due to obstacle detection errors, to reduce the need for operator attention in order to detect and intervene in situations where the robot makes incorrect decisions, and to reduce code complexity and the number of hand-tuned system parameters.

7 Summary and conclusions

This report presents and describes the proposed design, development and evaluation plan for the Intelligent Terrain Analysis and Tactical Support System (ITATSS) for Unmanned Ground Vehicles. The overall philosophy behind the architecture's design is centered on three primary points of emphasis:

1. Architectural components should use well-defined abstract interfaces to communicate with each other through a shared knowledge repository. This component-oriented approach will provide maximum flexibility in terms of selecting components to include in the architecture (from existing implemented systems), and extending the architecture in the future.
2. The primary application of ITATSS involves intense interaction with a human operator. Therefore, user-oriented design and development principles should be employed throughout the design, implementation, and testing processes.
3. Evaluation metrics and processes are key to creating and assessing the architecture and its components. We will employ evaluative processes to select implementations for architectural components, to identify the best approaches to integration, to design the most usable user interfaces, and to assess the overall effectiveness of the ultimate ITATSS implementation.

Following these principles, we have additionally identified and described a large number of existing systems that provide many of the various functions that must be integrated into ITATSS. We have analyzed existing capabilities in these systems, and identified potential augmentations that will enhance the effectiveness of the integrated system. We have also identified initial knowledge and behavior requirements for a knowledge-intensive decision support agent that will provide continual interaction and support for the robot operator in all phases of the operator's mission.

That ITATSS design and associated systems provide a high level of leverage for building an effective next-generation robotics system in a cost-effective manner. The implementation of this architecture is realistic, and the team members that prepared this report provide a unique combination of expertise to cover the necessary integration and development issues.

8 Appendix A

This appendix provides a description for the CFL and CNL languages used in MissionLab, which we propose to serve as part of the mission specification and abstract robot sensorimotor components for ITATSS.

The grammar for CDL is as follows:

```
Start           | Start DefineClass
                | Start DefineBP
                | Start InstAgent
                | Start InstGroup
                | Start InstSorA
                | Start DefineRobot
                | Start DefineOp
                | Start DefineArch
                | Start DefineType
                | Start InstCoord
                | Start InstBP
                | Start InstRobot
                | Start BindArch
                | Start DefineSandA
                | Start Agent
                | DefineClass
                | DefineBP
                | InstAgent
                | InstGroup
                | InstSorA
                | DefineRobot
                | DefineOp
                | DefineArch
                | DefineType
                | InstCoord
                | InstBP
                | InstRobot
                | BindArch
                | DefineSandA
                | Agent

Agent           | AGENT_NAME
                | ROBOT_NAME
                | GROUP_NAME
                | BP_NAME
                | RefRobot
                | RefGroup
                | RefCoord
                | RefSorA
                | RefBP
                | ClsRef
                | COORD_NAME

DefineArch      | defArch NAME ;

BindArch        | bindArch ARCH_NAME ;
                | bindArch NAME ;

TypeSetArch     | defType [ ARCH_NAME ]
                | defType [ NAME ]
                | defType

DefineType      | TypeSetArch NAME ;
                | TypeSetArch TYPE_NAME ;

DefineOp        | OpSetArch TYPE_NAME NAME FSA_STYLE EndDefop
                | OpSetArch TYPE_NAME NAME SELECT_STYLE EndDefop

EndDefop        | ( ParmDef ) ;
                | ( ) ;

OpSetArch       | defOp [ ARCH_NAME ]
                | defOp

DefineRobot     | RobotSetArch binds BP_CLASS NAME ( BPList ) ;
                | RobotSetArch binds BP_CLASS NAME ( ) ;

RobotSetArch    | defRobot [ ARCH_NAME ]
                | defRobot
```

```
BPList            | BPList , BPparmdef
                  | BPparmdef

BPparmdef         | sensor INLINE_NAME SENSOR_CLASS
                  | actuator INLINE_NAME ACTUATOR_CLASS

DefineBP          | StartBP ( ParmDef ) ;
                  | StartBP ( ) ;

StartBP           | defIBP TYPE_NAME NAME
                  | defIBP NAME NAME
                  | defOBP TYPE_NAME NAME
                  | defOBP NAME NAME
                  | defRBP NAME

SensorSetArch     | defSensor [ ARCH_NAME ]
                  | defSensor

StartSensor       | SensorSetArch binds BP_CLASS TYPE_NAME NAME
                  | SensorSetArch binds BP_CLASS NAME NAME

ActuatorSetArch   | defActuator [ ARCH_NAME ]
                  | defActuator

StartActuator     | ActuatorSetArch binds BP_CLASS TYPE_NAME NAME
                  | ActuatorSetArch binds BP_CLASS TYPE_NAME ACTUATOR_CLASS
                  | ActuatorSetArch binds BP_CLASS NAME

SandA             | StartSensor
                  | StartActuator

DefineSandA       | SandA ( ParmDef ) ;
                  | SandA ( ) ;

DefineClass       | StartClass ( ParmDef ) ;
                  | StartClass ( ) ;

AgentSetArch      | defAgent [ ARCH_NAME ]
                  | defAgent

StartClass        | AgentSetArch TYPE_NAME NAME
                  | AgentSetArch TYPE_NAME AGENT_CLASS

ParmDef           | ParmDef , AParm
                  | AParm

AParm             | TYPE_NAME NAME
                  | const TYPE_NAME NAME
                  | list TYPE_NAME NAME
                  | list const TYPE_NAME NAME

SorA              | ACTUATOR_NAME
                  | SENSOR_NAME

Loc               | ¡ NUMBER , NUMBER ¿
                  | ε

InstAgent         | instAgent Loc NAME from AGENT_CLASS ParmSet ;
                  | instAgent Loc NAME from SorA ParmSet ;
                  | instAgent Loc NAME from StartSorARef ParmSet ;
                  | instAgent Loc NAME from NAME ParmSet ;

ParmSet           | ( Glist )
                  | ( )

Glist             | Link
                  | Glist , Link

RobotParms        | ( RobotLinks )
                  | ( )

RobotLink         | Agent
                  | Link

RobotLinks        | RobotLink
                  | RobotLinks , RobotLink

Link              | LHS = Agent
                  | LHS = UP
```

```
                   | LHS = PU_INITIALIZER
                   | LHS = Rule
                   | LHS = INITIALIZER
                   | LHS =
                   | LHS = ( RobotLinks )

LHS                | PARM_NAME Loc
                   | PARM_NAME [ INDEX_NAME ] Loc
                   | PARM_NAME [ NAME ] Loc
                   | PARM_HEADER [ NAME ] Loc
                   | PARM_HEADER [ INDEX_NAME ] Loc
                   | PARM_HEADER
                   | PU_PARM_NAME Loc

MaybeAgent         | Agent
                   | ε

Rule               | if MaybeAgent goto INDEX_NAME
                   | if MaybeAgent goto NAME

MaybeName          | INLINE_NAME
                   | ε

StartClsRef        | MaybeName AGENT_CLASS

ClsRef             | StartClsRef ParmSet Loc

InstRobot          | StartRobotInst RobotParms

StartRobotInst     | instRobot Loc NAME from ROBOT_CLASS

InstCoord          | StartCoordInst ParmSet ;

StartCoordInst     | instOp Loc NAME from COORD_CLASS

RefRobot           | StartRefRobot RobotParms Loc

StartRefRobot      | MaybeName ROBOT_CLASS

RefCoord           | StartCoordRef ParmSet Loc

StartCoordRef      | MaybeName COORD_CLASS

RefSorA            | StartSorARef ParmSet Loc

SorAclass          | ACTUATOR_CLASS
                   | SENSOR_CLASS

StartSorARef       | MaybeName SorA Loc
                   | INLINE_NAME INLINE_NAME ACTUATOR_CLASS Loc
                   | INLINE_NAME INLINE_NAME SENSOR_CLASS Loc
                   | MaybeName SorAclass Loc

StartSorAInst      | instActuator Loc NAME from ACTUATOR_CLASS
                   | instSensor Loc NAME from SENSOR_CLASS

InstSorA           | StartSorAInst ParmSet ;

RefBP              | MaybeName BP_CLASS ParmSet Loc

InstBP             | instBP Loc NAME from BP_CLASS ParmSet ;

RefGroup           | [ AgentList ] Loc

InstGroup          | instGroup Loc NAME from [ AgentList ] ;

LinkorAgent        | Agent
                   | Link

AgentList          | LinkorAgent
                   | AgentList , LG
```

2. An actual CDL code snippet for an avoid-obstacle behavior appears below:

```
//*********************************************************
// Avoids obstacles
$Avoid_Obstacles_Mem:[
   %classes = {^},
   %avoid_obstacle_sphere = {^},
   %avoid_obstacle_safety_margin  = {^},
   %max_sensor_range = {^},
   %Decay_rate = {^},
   AVOID_OBSTACLES_MEM(
        %max_sensor_range = {^},
        sphere = {^Distances_10 %avoid_obstacle_sphere},
        safety_margin = {^Distances_10 %avoid_obstacle_safety_margin },
        %classes = {^},
        readings = $ListOfObstacles,
        cur_pos = $RobotLocation,
        decay_rate = {^Range_01 %Decay_rate}
   )<393,26>|avoid obstacles with memory|
]<66,36>|Move the robot away from objects not listed as OK|
//*********************************************************
```

3. CNL Code for Avoid-static-obstacles:

```
procedure Vector AVOID_STATIC_OBSTACLES with
    double sphere;
    double safety_margin;
    obs_array readings;
once
header
body
    VECTOR_CLEAR(output);

    for(int i=0; i<readings.size; i++)
    {
        double c_to_c_dist = len_2d(readings.val[i].center);
        double radius = readings.val[i].r + safety_margin;
        double mag = 0;

        if (c_to_c_dist ≤ radius )
        {
            // if within safety margin generate big vector
            mag = INFINITY;
        }
        else if (c_to_c_dist ≤ radius + sphere )
        {
            // generate fraction (0...1) how far are in linear zone.
            mag = (sphere - (c_to_c_dist - radius)) / sphere;
        }
        // otherwise, outside obstacle's sphere of influence, so ignore it

        if (mag ≠ 0)
        {
            // create a unit vector along the direction of repulsion
            Vector repuls;
            repuls.x = -readings.val[i].center.x;
            repuls.y = -readings.val[i].center.y;
            unit_2d(repuls);

            // Set its strength to the magnitude selected
            mult_2d(repuls, mag);

            // Add it to the running sum
            plus_2d(output, repuls);
        }
    }
pend
```

9 Appendix B

The Compact Terrain Database library, libctdb, is used by an OTB application to access elevation, soil type, and feature data of a SIMNET database. Terrain databases are compiled from S1000 source (or other source formats) into libctdb format. OTB applications use libctdb to load this database into memory, and then use libctdb functions to access the data therein. With format 5, the ability to tile large databases using a global coordinate system has been added.

Libctdb functions include:

- Reading the database into memory or cache
- Maintaining useful information about the database, such as its size, minimum and maximum elevation, and UTM zone, northing and easting or GCS cell id (its location on the planet)
- Point elevation lookup
- Elevation lookup along a line segment (find high ground, find terrain profile, etc.)
- Soil type lookup
- Vehicle placement (rotation matrix generation)
- Intervisibility calculation (including terrain and vehicle blockage)
- Radar clutter calculation
- Generating graphic representation of the terrain such as contour maps and hypsometric maps, in real time

The CTDB header format includes three flags which indicate the terrain database profile. There are two types of terrain databases currently being generated. Version 1 databases are mostly gridded, where elevation posts are stored at regular distances and triangulated. These databases may have more detail added within the triangles using microterrain triangles. Some newer databases are generated completely from microterrain, in a Triangulated Irregular Network (TIN). With CTDB format 4, the representation was updated to store this type of database more efficiently.

In addition, there are three GCS tiling modes supported. These modes, specified by the field in the ctdb database header, are:

1. CTDB_MODE_SIMNET : A SIMNET style database based on a UTM projection. SIMNET databases do not use the Global Coordinate System, and only one SIMNET database may be used by an application at any given time.
2. CTDB_MODE_SINGLE_CELL : A single cell database uses Global Coordinate System coordinates (i.e., no projection is performed, so the earth's curvature is accurately represented), but does not break large databases up into tiles (cells). Only one single cell mode database may be used by an application at any given time.
3. CTDB_MODE_MULTI_CELL : A multi-cell database uses the Global Coordinate System and breaks up large databases into cells. With support from libGCS and libWorld, multiple cells, each of which is an independent database, can be loaded and used as a single virtual database.

Global coordinate system design philosophy

The GCS Design Philosophy breaks the world up into cells, which are generally one degree of latitude by one degree of longitude. Each cell in the playbox then has an associated terrain database (note that it should be possible, in the future, to extend the system to allow multiple ctdbs per cell) which covers the entire area of the cell. Note that each such database is a complete ctdb, and can be used independently if desired.

The mapping from cells to ctdbs is managed by libGCS; conversely, the libctdb compiler stores the cell in which each ctdb falls in that ctdb's data. This information is used to determine the frame of reference in which a given ctdb's data is stored. In addition, it allows us to greatly limit changes to the previous API, since nearly all API routines already took a ctdb pointer as an argument, and ctdb pointers can be used to specify a frame of reference for coordinates passed along with them (i.e., this saved us from having to add a cell id to each API routine that took coordinate arguments). In general, we have adhered to the following rules in attempting to maintain the previous API:

- All coordinate arguments passed to a function are expected to be in the frame of reference of the accompanying ctdb pointer, as explained above.
- All coordinate return values will be returned in the frame of reference of the passed ctdb.
- If a query requests information that cannot be obtained from the passed ctdb, libctdb will determine the proper ctdb to use (by querying libGCS), call itself recursively, convert the result if necessary, and return the answer to the query.

To understand the implications of these rules, consider the following two examples:

An elevation lookup query is made for a point outside the database in which the query is made (e.g. the database bounds are (0,0) to (100000, 100000), and the point queried is (-15000, 20000)). libctdb will, upon discovering that the queried point is not in this database, do the following:

Query libGCS to determine the cell in which the point falls.

Query libGCS to find the ctdb registered to that cell. If no such database is registered, a default behavior is invoked. Currently, elevation lookup will snap the point to the nearest point on the database in which the call was made and return the elevation there.

Convert the point's coordinates to the new cell's frame of reference.

Perform a new elevation lookup using the new ctdb and converted coordinates.

Convert the result into the original frame of reference (remember that the z-value assumes a frame of reference).

Return the converted elevation.

An intervisibility test is requested between points in two different cells. In order to make this test, the application must determine the coordinates of both points in a single frame of reference, then call the ctdb intervisibility function with the ctdb for that frame of reference. libctdb will then segment the ray at cell boundaries, perform intervisibility calculations in each relevant cell, and combine the results.

Contour line generation

The libctdb library provides the ability to generate contour lines at any interval. It can prepare these contour lines for display using two different methods:

- Periodic sampling to generate a raster image of the contours (using the X-windows XY-Bitmap image format).
- Analysis of terrain to find the contour lines which cross each polygon.

Although either method can be used at any map scale, the periodic sampling method is most effective when the scale is very high (1:200,000 or more).

Periodic sampling

The algorithm to determine contour lines for each cell database via sampling is quite simple. Keep in mind that in GCS modes, we must use altitude above mean sea level to avoid non-intuitive effects, such as the generation of contour lines in the middle of the ocean. The algorithm is as follows:

- Create a 2 dimensional array.
- Sample each point in the area to be mapped, such that there is 1 sample point per pixel in each dimension.
- Find the elevation above sea level at each sample point and round it down to the nearest contour line (*e.g.*, if contours are being generated every 5 meters, and the elevation at the point is 19.5, the rounded elevation is 15). Store these elevations in the 2D array.
- For each location in the elevation array, plot a point if the elevation of that location is larger than the elevation of any surrounding locations.

Analytic generation

The analytic method is more complicated; however, it runs much faster than the sampling method at lower map scales (since the number of polygons tested will be lower than the number of pixels on the screen), and hence is more generally useful. The lines for each cell database are generated in two passes. First lines are generated for the regular elevation grid, then lines are generated for TIN polygons and microterrain.

This general method is used for each polygon encountered:

- Find the minimum and maximum elevations of the polygon.
- Starting with the contour elevation below highest elevation, and proceeding through the contour elevation above lowest elevation, determine where the contour elevation intersects the edges of the polygon. For each pair of intersections, store a line segment connecting them.

This algorithm is simple, and there are a wide variety of options to use for implementation. Specific optimizations include:

Input line buffering

Each set of four points in the regular grid contributes to many different triangles. Hence, a significant reduction in the number of elevation lookups can be achieved simply by saving those lookups already performed. This is done in both dimensions, by keeping two horizontal buffers -- as one is being filled, the other is being used. The buffers are filled as information is needed (rather than all at once) which prevents stalling of the RISC pipeline (arithmetic operations continue while the memory fetch completes).

Elevation basis adjustment

Since elevations are not stored in meters, but rather in a more compact fixed-point format, they must be converted to meters before they can be used. But rather than converting to meters, the fixed point basis is adjusted at the beginning of the routine to convert directly to the interval of contouring (*e.g.*, units of 5 meters). This simplifies the rest of the routine, which can assume contours every 1.0 units.

Exploitation of grid orientation

Since the grid is oriented with the Cartesian plane, determining the points where contour elevations intersect polygon edges can be done using simple parametric methods in one variable.

Exploitation of triangles

Since each polygon is a triangle, no calculation is needed to determine which contour-edge intersections to join into line segments.

Nudging Contour Elevations

Since the original source data for the surface polygons (DMA, the Defense Mapping Agency) fixes heights at meter resolution, strange contours can occur when finding contours at meter resolutions (contours often trace edges in the grid, for example). To make more natural looking contour lines, all the elevations are nudged up a little (*e.g.*, 10 meter intervals would test contours at 10.00001, 20.00001, 30.00001 and so on).

Abstract Features

Abstract features are stored after the topology. To allow efficient searching, the features are stored in a quad tree data structure. The locus of each quad node and the resolution of quad tree subdivision are all implied by the underlying data structures. The quad tree does not have to be uniformly subdivided. Each abstract feature can have extra data associated with it. This data is stored in a variable-length union. The `type_code` of the feature acts as a tag. The `canopy` contains a flag indicating if the canopy is to be treated as impenetrable. The `soil_defrag` contains an indication of soil type (a soil table index and an index into that table), as well as a cardinal number indicating layering for the purposes of map drawing (for example, an deep lake surrounded by fordable marsh). The `steep_slope` contains a measure of terrain slope, in degrees. The `label` contains a null-terminated character string. The `offroad` contains an index identifying the number of the topological edge which contains the offroad feature

Visibility around vehicles

The calculation of visibility with respect to intervening vehicles is done separately from the calculation with respect to intervening terrain. The interference generated by vehicles is collected into a single representative raster which is then combined with the raster from terrain data to compute a net result.

The algorithm used assumes that all vehicles are equally likely to interfere with visibility. In applications where large numbers of far away vehicles would occur in a simple table, a filtration can be made by the application prior to intervisibility calculation (such as collecting only nearby vehicles for testing from a position-based vehicle table). The algorithm simply runs through all the vehicles provided, and determines which (if any) block visibility, and to what extent.

The mechanics of determining the blockage from each vehicle is a combination of two routines. First, the vehicle can be treated as a rectangle which always faces the viewer, the calculation of whether the vehicle blocks visibility is identical to that used for trees (which are also just rectangles which face the viewer.

Second, if the interfering vehicle does block any of the target, a raster is generated which represents the extent of the blockage. This is done as follows:

Determine the total amount of target width (projected back to the point of intersection with the obstruction) which is blocked, except do not clip to maximums of `2*radius` and `target width`. Call this quantity `A`..

If `A` is larger than the target size at this point, the target is completely blocked.

If the center of the obstruction is to the right of the line of sight to the center of the target, the left side of the blockage will start at `larger of[(width - amount_obstructed), 0.0]`, and the right side will start at `smaller of[(left + amount_obstructed), width]`.

If the center of the obstruction is to the left of this line, the same rules apply, but are reversed.

Use these extents as the left and right bounds of the raster. The top and bottom extents are found by comparing the elevation of the high and low sight lines to the elevation of the top and bottom of the blocking object.

The rasters generated for each intervening vehicle are combined, resulting in one vehicle-blockage raster.

Intervisibility for terrain elements

The terrain element intervisibility engine performs the same function as the feature and grid post intervisibility engine, but looks only for TIN polygons along the line of sight in the current database. The engine will invoke a passed function on every terrain element edge intersected by the line of sight.

One of the benefits of using terrain elements is that the stored topology information allows for a very efficient traversal of the terrain elements intersecting a line of sight. Libctdb aims to make maximal use of this.

In principle, it should be possible to use the stored adjacency information and an initial TE to traverse the entire line of sight in a given database, finding all intersected TE's. However, since terrain element indices are only unique within a given patch, simple adjacency storage would force us to stop and find an initial TE every time we entered a new patch. We avoid this problem using Smart-Edge adjacency storage. Because the coordinate system used is patch-dependent, we process TE's one patch at a time, but retain adjacency information between patches.

Water characteristics

The water characteristics table stored in the CTDB data structure maintains a list of unique sets of attributes of water surface polygons that occur within the database. Water polygons then store indices into this table to define their characteristics. This scheme allows for efficient storage of water characteristics when these characteristics do not vary frequently (as one would expect).

Soil tables

Until CTDB format 5, there was a long standing limitation to the number of soil types that any given ctdb database could store. This limitation was remedied by the introduction of multiple and variable soil tables. In CTDB format 5, the number of bits reserved for soils was either 4 (soil defrag, posts, microterrain features and linear features) or 8 (terrain elements). In an effort to minimize the scope of the soilmap change and to increase the number of soils to a maximum of 16 soils per gridded patch or 256 soils for a tinned patch, a level of indirection through the patch was added. The soil index in each internal ctdb feature, abstract and post structure was defined as an index into a patch local soil table. Since a majority of the soil patches could probably use the same soil table, an index into a master list of soil tables was used to gain access to the proper local soil table information.

10 References

Ali, K., & Arkin, R.C. (2000). Multiagent teleautonomous behavioral control. *Machine Intelligence and Robotic Control, 1*(2), 3-10.

Arkin, R.C., Collins, T.R., & Endo, T. (1999). Tactical mobile robot mission specification and execution. Mobile Robots XIV, 150-163. Boston, MA.

Arkin, R.C., & Diaz, J. (2002). Line-of-sight constrained exploration for reactive multiagent robotic teams". *7th International Workshop on Advanced Motion Control, AMC'02*, 455-461. Maribor, Slovenia.

Barnett, J.S. and Meliza, L. (2003). Automation integration: Comparing flightdeck automation and U.S. Army digitization. I Proceedings of 2003 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC). Orlando, FL.

Brooks, R. A. (1987). Intelligence without representation. *Artificial Intelligence*. 47, pp. 139-159.

Balch, T., & Arkin, R.C. (1998). Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation, 14*(6), 926-939.

Card, S., Moran, T. & Newell, A. (1983). The Psychology of Human-Computer Interaction. Hillsdale, NJ: Erlbaum. Castelnovi, M., Arkin, R.C., and Collins, T.R. (2005). Reactive speed control system based on terrain roughness detection. Submitted to ICRA-05, Barcelona, April, 2005.

Castelnovi, A., Arkin, R., and Collins, T. (2005). Reactive Speed Control System Based on Terrain Roughness Detection. To appear in *Proceedings of IEEE International Conference on Robotics and Automation*. Barcelona, Spain.

Collins, T.R., Arkin, R.C., Cramer, M.J., & Endo, Y. (2000). Field results for tactical mobile robot missions. *Unmanned Systems*. Orlando, FL.

Drewes, P. (2003). Demonstration of a systems architecture for live, virtual, and constructive UGV operation. Proceedings of AUVSI's Unmanned Systems Symposium. Baltimore, MD.

Endo, Y., MacKenzie, D., and Arkin, R.C., "Usability Evaluation of High-level User Assistance for Robot Mission Specification'"*, IEEE Transactions on Systems, Man, and Cybernetics*, Vol.34, No.2, May 2004.

Fields, M. (1999). Modifying ModSAF terrain databases to support the evaluation of small weapons platforms in tactical scenarios. ARL Technical Report ARL-TR-1996.

Fields, M. (2001). Representing ground robotic systems in battlefield simulations.

Fong, T., Thorpe, C., and Baur, C. (2001). Advanced interfaces for vehicle teleoperation: collaborative control, sensor fusion displays, and remote driving tools. *Autonomous Robots*, 11(1), pp. 75-85.

Gerkey, B., Vaughan, R., & Howard, A. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317-23, Coimbra, Portugal.

Halme, A., and Suomela, J. (2000). Tele-existence techniques of heavy work vehicles. IEEE Workshop on Teleoperation, *IEEE International Conference on Robotics and Automation*, San Francisco, CA.

Jones, R. M. (1999). Graphical visualization of situational awareness and mental state for intelligent computer-generated forces. *Proceedings of the Eighth Conference on Computer Generated Forces and Behavioral Representation*, 219–222. Orlando, FL .

Jones, R. M., & Kenny, P. G. (2000). Lessons learned from integrating STOW technology into an operational naval environment. *Proceedings of the 2000 Spring Simulation Interoperability Workshop*, Orlando, FL.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. G., & Koss, F. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20*(1), 27-41.

Kluge, K., & Morgenthaler, M. (2004). Multi-horizon reactive and deliberative path planning for autonomous cross-country navigation". To appear in *Rough Terrain Mobility*, SPIE Proceedings Vol. 5422. Orlando, FL.

Krotkov, E. (1996). Safeguarded Teleoperation for Lunar Rovers: From Human Factors to Field Trials. *Proc. IEEE Planetary Rover Technology and Systems Workshop*, Minneapolis, MN.

Laird, J. E., Newell, A., & Rosenbloom, P. S.: *Soar: An architecture for general intelligence*. Artificial Intelligence, Vol. 33, pp. 1-64, 1987.

MacKenzie, D. and Arkin, R.C. (1993). Formal specification for behavior-based mobile robots. *Mobile Robots VIII,* pp. 94-104. Boston, MA.

MacKenzie, D., Arkin, R.C., & Cameron, R. (1997). Multiagent mission specification and execution. *Autonomous Robots, 4*(1), 29-52.

MacKenzie, D, & Arkin, R. (1998). Evaluating the usability of robot programming toolsets. *International Journal of Robotics Research, 4*(7), 381-401.

Morgenthaler, M. "UGV Mission Planning," Reconnaissance, Surveillance, and Target Acquisition for Unmanned Ground Vehicles: Providing Surveillance 'Eyes' for an Autonomous Vehicle, O. Firshein and T. Strat, ed, pp 109-128, Morgan Kaufmann, 1997.

Morgenthaler, M., Dickinson, A., & Glass, B. (2000). XUV/DemoIII multi-vehicle operator control unit. *Unmanned Ground Vehicle Technology II,* SPIE Proceedings Vol.4024. Orlando, FL.

Musliner, D. (ed.), (1999). *Working Notes of the AAAI Spring Symposium on Adjustable Autonomy,* AAAI Press, Stanford, CA.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Ritter, F. E., Jones, R. M., & Baxter, G. D. (1999). Reusable models and graphical interfaces: Realising the potential of a unified theory of cognition. *Mind modeling: A cognitive science approach to reasoning, learning and discovery,* 83–109. Lengerich, Germany: Pabst Scientific.

Sheridan, T. (1992). *Telerobotics, Automation, and Human Supervisory Control,* MIT Press.

Taylor, G. E, Jones, R. M., Goldstein, M., Frederiksen, R., & Wray, R. E. (2002). VISTA: A generic toolkit for visualizing agent behavior. *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavioral Representation*, 157-167. Orlando, FL.

Wood, S.D., Zaientz, J.D., Beard, J., Fredriksen, R., & Huber, M. (2003). CIANC3: An agent-based intelligent interface for future combat systems command and control. *Proceedings of the 2003 Conference on Behavior Representation in Modeling and Simulation (BRIMS).* Scottsdale, AZ.

Wray, R. E., & Laird, J. E. (2003). Variability in human behavior modeling for military simulations. *Proceedings of the 2003 Conference on Behavior Representation in Modeling and Simulation.* Scottsdale, AZ.